Scalable and Efficient Distributed Self-Healing with Self-Optimization Features in fixed IP Networks

vorgelegt von
Diplom-Informatiker
Nikolay Tcholtchev

Technische Universität Berlin
Fakultät Elektrotechnik und Informatik

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
Dr.-Ing.

genehmigte Dissertation

Vorsitzender: Prof. Dr. Giuseppe Caire
Berichterin: Prof. Dr. -Ing. Ina Schieferdecker
Berichter: Prof. Dr. Axel Küpper
Berichter: Prof. Dr. habil. Michael Menth

Tag der wissenschaftlichen Aussprache: 17.07.2019

Berlin 2019
D 83

Abstract

The Internet is continuously gaining importance in our society. Indeed, the Internet is slowly turning into the backbone of the modern world, having impact on all possible aspects, such as politics, communication, intercultural exchange, and emergency services, to give some examples. As these aspects are developing, the technical infrastructure around the Internet's core protocol - IP (Internet Protocol) - is increasingly exposed to various challenges. One of these challenges is given by the requirement for sophisticated resilience mechanisms that can guarantee the robustness of the IP infrastructure in case of faults, failures, and natural disasters. This is of paramount importance for two reasons. First, Internet type of networks are to be deployed in the area of safety critical systems, such as emergency services (e.g. VoIP emergency services) that will be used in the course of network recovery after a disaster event (e.g. flooding, acts of terrorism, etc.). Secondly, the more robust the infrastructure, the higher the services' availability and correspondingly the revenue for the service provider. Thus, resilience and robustness turn out also to be desired features beyond the above mentioned safety use case.

This dissertation aims to develop a new architectural framework for improving the resilience of network nodes in fixed IP network infrastructures, i.e. IP networks without any mobility and continuously changing physical topology. The current thesis approaches the topic of resilience from two different perspectives. First, it is recognized that resilient self-healing mechanisms are already embedded inside diverse network protocols, as well as in applications and services running on top of a fixed IP network. Secondly, the importance of network and systems management processes for the availability of the network and IT infrastructure is also analyzed. This leads to the identification of a gap between the resilient features which are intrinsically embedded inside the protocols and applications, on one hand, and the network and systems management processes, on the other hand. This gap is constituted by the lack of a framework that runs on top of the protocols and applications and manages them with respect to incidents, thereby automating aspects of the established management standards. In addition, this framework is meant to serve as a layer between the network/system's administrator and the networked infrastructure. That is, on one hand, the framework is configured and provided with knowledge by the human experts tweaking and improving the system. On the other hand, the framework is designed to escalate faulty conditions, which it is not able to resolve, to the operations personnel, such that responsive managerial actions can be initiated.

The architectural framework consists of software components that operate in a distributed manner inside the nodes of the networked system in question. These software components are able to proactively and reactively respond to faulty conditions, i.e. on one hand failures are predicted and avoided, and on the other hand, an automatic response to already existing faulty conditions is realized. Correspondingly, existing mechanisms to realize these processes are evaluated, and where required new algorithms are developed for the proposed framework - for instance scalable Markov Chain based Fault-Isolation or efficient self-optimization and action synchronization

based on mathematical optimization and machine learning techniques.

To evaluate the concepts and mechanisms, a number of case studies are executed. Additionally, the scalability and overhead (e.g. memory consumption) of the proposed framework are evaluated. Moreover, the required interfaces between the network administrator and the self-healing system are investigated and defined accordingly. It is ensured that the network administrator has complete control over the network, can oversee the automated processes, and can even intervene if required. Furthermore, the framework and algorithms are designed in a way that enables the realization of real-time self-healing whereby the reaction strategy is always optimized such that key performance indicators of the networked system are improved.

Zusammenfassung

Das Internet gewinnt in unserer Gesellschaft kontinuierlich an Bedeutung. In der Tat verwandelt sich das Internet in das Rückgrat der modernen Welt, mit Auswirkungen auf zahlreiche Aspekte wie Politik, Kommunikation, interkultureller Austausch und Notdienste, um einige Beispiele zu nennen. Die Weiterentwicklung dieser Bereiche stellt die technische Struktur des Internets - insbesondere das Internet Protokoll (IP) - zunehmend vor verschiedene Herausforderungen. Aus diesen Herausforderungen ergibt sich unter anderem die Notwendigkeit für skalierbare und effiziente Selbstheilungsmechanismen, die in den darunterliegenden Netzen zu integrieren sind. Das Ziel dieser Selbstheilungsmechanismen besteht in der Gewährleistung von Robustheit und Widerstandsfähigkeit der IP-Infrastruktur bei Störungen, Angriffen und Naturkatastrophen. Dies ist aus zwei Gründen von großer Bedeutung. Erstens, IP-basierte Netzwerke werden zunehmend im Bereich der sicherheitskritischen Systeme eingesetzt, wie zum Beispiel VoIP-Notdienste, die im Verlauf der Netzwiederherstellung nach einem Katastrophenereignis (zum Beispiel Überschwemmung, Terrorakte) eingesetzt werden. Zweitens, je robuster und widerstandsfähiger die Infrastruktur desto höher ist die Verfügbarkeit der Dienste und entsprechend die Einnahmen für den Netzwerk- bzw. Dienstanbieter. Dementsprechend erweisen sich die Widerstandsfähigkeit und Robustheit von IP-Netzen als notwendige Eigenschaften, auch über die Anforderungen von dem oben erwähnten Einsatzfeld innerhalb von sicherheitskritischen Systemen hinaus.

Diese Dissertation setzt sich als Ziel, eine Softwarearchitektur zu entwickeln, durch die eine Verbesserung der Widerstandsfähigkeit von Netzwerkknoten in festen IP-Netzwerkinfrastrukturen realisiert werden kann, das heißt IP-Netzwerke ohne Mobilität und kontinuierlich wechselnde physikalische Topologie. Die Dissertation betrachtet das Thema der Widerstandsfähigkeit aus zwei verschiedenen Perspektiven. Zuerst wird analysiert und festgestellt, dass dynamische Selbstheilungsmechanismen bereits in diversen Netzwerkprotokollen sowie in Anwendungen und Diensten, die in einem festen IP-Netzwerk laufen, eingebettet sind. Im Anschluss daran wird die Bedeutung von etablierten Netzwerk- und Systemmanagementprozessen für die Verfügbarkeit von Netzwerk- und IT-Infrastruktur analysiert. Dies führt zur Identifizierung einer Lücke zwischen den dynamischen Selbstheilungs- und Robustheitsmechanismen, die einerseits in den Netzwerkprotokollen und Anwendungen/Diensten und andererseits in den etablierten Netzwerk- und Systemmanagementprozessen eingebettet/integriert sind. Die Lücke besteht darin, dass es an einer Softwarearchitektur fehlt, die on-top der Netzwerkprotokolle und Anwendungen läuft, diese in Bezug auf Vorfälle verwaltet und damit die Aspekte der etablierten Management-Standards im Sinne eines Selbstheilungsprozesses automatisiert. Darüber hinaus soll dieses Framework als Zwischenschicht zwischen dem Netzwerk-/Systemadministrator und der Netzwerk-/Dienstinfrastruktur dienen. Das heißt, einerseits wird das Framework konfiguriert und mit Kenntnissen der Experten/Administratoren ausgestattet, die das Netzwerk/System ausbauen, pflegen und detailiert kennen. Andererseits ist die anvisierte Softwarearchitektur so konzipiert, dass sie Störungen, die sie nicht lösen kann, an die Mitarbeiter des Network Operations Center (NOC) eskaliert. Die

NOC-Mitarbeiter sind anschließend in der Lage entprechende Aktitvitäten einzuleiten, um das bestehende Netwerkproblem zu behandeln, das von dem anvisierten Framework nicht gelöst werden konnte.

Die vorgeschlagene Architektur besteht aus Softwarekomponenten, die in verteilter Weise innerhalb der Knoten des betreffenden vernetzten Systems arbeiten. Diese Softwarekomponenten sind in der Lage, proaktiv und reaktiv auf fehlerhafte Zustände zu reagieren, das heißt einerseits werden Ausfälle vorhergesagt und vermieden, während andererseits eine automatische Antwort auf bereits vorhandene fehlerhafte Bedingungen realisiert wird. Entsprechend werden bestehende Mechanismen zur Realisierung dieser Prozesse evaluiert, und wo es erforderlich ist, werden neue Algorithmen für die anvisierte Architektur entwickelt - zum Beispiel skalierbare Markov-Ketten basierte *Fault-Isolation* oder effiziente Selbstoptimierung und Action-Synchronisation basierend auf mathematischer Optimierung und Techniken aus dem Bereich des Maschinellen Lernens.

Zur Bewertung der vorgeschlagenen Konzepte und Mechanismen werden eine Reihe von Fallstudien durchgeführt. Zusätzlich werden die Skalierbarkeit und der Overhead (z.B. Speicherverbrauch) der vorgeschlagenen Softwarearchitektur ausgewertet. Darüber hinaus werden die benötigten Schnittstellen zwischen dem Netzwerkadministrator und dem Selbstheilungssystem entsprechend untersucht und definiert. Das Gesamtkonzept sieht vor, dass der Netzwerkadministrator die volle Kontrolle über das Netzwerk behält, die automatisierten Prozesse überwachen kann und bei Bedarf jederzeit eingreifen kann. Darüber hinaus sind das Framework und die zugehörigen Algorithmen so konzipiert, dass die Realisierung von Echtzeit-Selbstheilung möglich ist. Dabei wird die Reaktionsstrategie des Frameworks immer so optimiert, dass wichtige Leistungsindikatoren (Key Performance Indicators (KPIs)) des vernetzten Systems verbessert werden.

Acknowledgements

To my family and to my late father.

First of all I would like to express my special thanks and gratitude to Prof. Dr. Ina Schieferdecker, who supervised my thesis and with whom we worked on a number of exciting topics, including Quality Assurance and Testing for communication based systems, Urban Data Platforms, Autonomic Network Management, and ICT Reference Architectures for Smart Cities. Being my direct Research Leader at the MOTION[1] Competence Center and at the System Quality Center (SQC) of Fraunhofer FOKUS[2], as well as later in the position of Institute Director, she gave me the possibility and the needed advice that helped enormously to develop myself as a researcher and improve my results and publications. I highly appreciate her support, and the opportunity she gave me to learn and develop scientifically in the field of ICT and telecommunications.

During the past years at Fraunhofer FOKUS, I had the chance to work with a large number of colleagues who supported my PhD motivation and helped me to master a number of complex projects in a challenging scientific and industrial context. All of these colleagues influenced me and enabled my personality to develop in a way that made it possible to finalize the scientific activities at the heart of the dissertation endeavor. The following list is by far not exhaustive, however I would like to thank Tom Ritter, Andreas Hoffmann, Faruk Catal, Rene Richter, Philipp Lämmel, Robert Scholz, Evanela Lapi, Florian Marienfeld, Arun Prakash, Steven Ulrich, Wojciech Konitzer, Alexej Starschenko, Max Bureck, Thekla Heuer and Ranganai Chaparadza.

Last but not least, I would like to express my gratitude to all my friends who encouraged and motivated me to continuing and finishing the dissertation.

Berlin, February, 2018

[1] Modeling and Testing for System and Service Solutions
[2] Fraunhofer Institute for Open Communication Systems

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is undoubtedly one of the major innovations of the past decades. It significantly changed people's lives across the planet and shaped new economic and social dimensions. The merge of different sub-networks into a common self-organizing network of networks, such as the Internet, allows for almost unlimited and open communication between arbitrary parties in the world.

The forms of communication over the Internet and the resulting demands are constantly changing. It began, among others, with various services for the exchange of messages and data, e.g. e-mail (SMTP, POP, IMAP, ...), transfer of files (FTP, UUCP), distributed file systems (NFS, AFS, ...), WWW (Apache, CGI, HTTP, HTTPS, ...) and chat (IRC, ICQ, ...). Developments of recent decades include the communication concepts that have emerged in relation to Web 2.0, such as social networking (facebook, LinkedIn, twitter ...), blog services, video-sharing platforms (YouTube, ...), and knowledge bases such as Wikipedia. In addition, the rapid emergence of VoIP (Voice over IP) services and applications, as well as the importance gaining idea of service-oriented software (e.g. SOAP and WSDL), turn the Internet into a vital communication medium that gains more and more significance as an industrial and social environment. Some latest developments are given by concepts and deployments such as *"Internet of Things* - the linking of any objects like sensors and kitchenware, *"Cloud Computing"* - software, computing/storage capacity, and network infrastructure as a shared (mostly) virtualized service, *"Mobile Internet"* - access to the Internet from anywhere at any time, and *"Interplanetary Internet"* - a delay tolerant version of an Internet infrastructure that facilitates communication in space.

All the above present and future developments inevitably complicate additionally the already challenging network management processes. Among others, the robustness and reliability of the network infrastructure including routers, switches, hubs, gateways, links and network interface cards is exposed at risk, since the resolution of a network failure (caused for example by a configuration problem in a router) becomes more complex and expensive. The variety of technologies and network protocols (e.g. OSPF, BGP, IPv4/v6, MPLS, ATM, Ethernet, Frame Relay, PPP, ...), the various hardware platforms, and software implementations make the manual troubleshooting of network problems a difficult to oversee task, which can usually only be accomplished by highly qualified professionals.

The above considerations clearly indicate that there is a need to introduce and embed the knowledge of the limited number of highly qualified professionals - coping with the network and systems management challenge - into the managed network elements, in order to achieve a certain degree

of automation with respect to problems of the network, which can be solved by a dedicated piece of software (e.g. a software agent based framework). Indeed, it should be seen as a fundamental need to conceptualize required components, which have the capability to handle emerging operational problems in a distributed manner and increase the robustness and resilience for a fixed IP network infrastructure. Such fixed IP networks are found in different parts of the Internet, such as Autonomous Systems (AS) in the Internet backbone, various local area networks (Local Area Network - LAN) like university and campus networks, corporate networks (enterprise networks) and home networks.

The required (software) components - to be designed in the scope of this thesis - should be able to proactively and reactively respond to faulty conditions, i.e. on one hand failures should be predicted and avoided, and on the other hand, an automatic response to already existing faulty conditions should be realized. Correspondingly, algorithms to realize these processes are required. Therefore existing mechanisms must be evaluated, and where required new algorithms have to be specified and developed for the purposes of the emerging software components. Furthermore, it must be ensured that the proposed software components operate in a scalable manner thereby keeping the induced overhead (memory consumption, CPU utilization, response times, ...) minimal. Indeed, it is of paramount importance to keep the network administrator in the loop, whilst in parallel improving the QoS (Quality of Service) of the network by automatic proactive and reactive operations within the network devices. Hence, by defining suitable interfaces between the operations personnel and the self-healing system, it has to be ensured that the humans retain complete control over the network/system in question, and can oversee the automated processes - however without being intensively required to manually execute traditional Fault-Management and troubleshooting tasks. These high level thoughts are further motivated and brought to a specific problem statement within the coming sections.

1.1 Motivation

The Internet impacts the society mainly through the applications and services running on top of the global network. From the communication networks perspective - different technologies, like ATM (Asynchronous Transfer Mode), Frame Relay, X.25 and Token Ring - have been deployed and integrated to interoperate. In order to facilitate the interplay of different networking technologies, the Internet Protocol (IP) has been standardized in the eighties as the gluing technology to enable the interoperability and integration of different networks. Subsequently, all applications and services can rely on the existence of an IP layer that is able to carry their messages across the world, regardless of the underlying communication network substrate. To facilitate the understanding and development of Internet protocols and services, two reference models have been mainly applied that create frameworks for discussions and further evolution. These frameworks [1] include the ISO Open Systems Interconnection (OSI) model coming from traditional telecommunications, and the TCP/IP model that emerged with the development of the Internet. Both models aim at describing communication networks (including the Internet) in the framework of layers that comprise all aspects of communication, starting from the physical network access, and going up to the level of applications and services.

As IP was facilitating the growth of the Internet throughout the years, there was a clear need for increased robustness of the Internet fixed infrastructure, including Local and Wide Area Networks

[1] An illustration of both models and related communication technologies is provided in Figure A.1 within the appendix of this thesis.

(LAN and WAN). On one hand, within the network protocols, this was achieved by developing self-healing stacks [Gre95], embedding recovery and restoration schemes inside the protocols (e.g. SONET, MPLS and FDDI), providing mechanisms for failure/error notification such as ICMP, and enabling the transport layer to react to issues like network congestion, e.g. as realized in TCP. On the other hand, the control plane/layer was extended by routing protocols that are able to automatically recover from link or node failures, and reroute the affected traffic over an alternative path. Typical protocols of that type are given by OSPF (Open Shortest Path First) and RIP (Routing Information Protocol).

From the network and systems management perspective: Different protocols were specified within the management plane/layer to facilitate fast and efficient troubleshooting and network management. In that context, traditional network management is understood according to the ITU-T TMN management standard that defines the FCAPS framework - Fault-, Configuration-, Accounting-, Performance-, and Security-Management. This includes protocols such as SNMP (Simple Network Management Protocol) which facilitate the monitoring and the management of the network infrastructure. The output of these protocols is further integrated into Network Management systems (e.g. Open View [TZ04]). Furthermore, the adoption of scripting has automated many of the network management tasks. According to dos Santos et. al. [dSBC+10], *"Today, it is not uncommon to find situations where management information ends up being integrated through scripts written by network administrators in a domestic ad hoc way."*, even being it any type of modern Software Defined Networking scripts (including Network Functions Virtualization). However, this scripting solution is by far not optimal and brings a variety of pitfalls for the goal of building a sustainable and evolvable network management, and correspondingly Fault-Management and troubleshooting procedures.

- First of all, the network operations personnel needs to be highly skilled and have both network management knowledge and programming knowledge.
- Secondly, a script programming solution may be extremely difficult to understand and maintain by others than its developers.
- This may lead to a situation where know-how is lost leading to disruption in the network management processes and to a decreased reliability of the network and IT services provided by an organization.

In order to resolve such issues, a wave of initiatives was started during the past decade that tried to introduce self-management capabilities inside or close to the network and services' architectures. Some of these initiatives looked at the management plane/layer as an unnatural add-on for the TCP/IP protocol suite, and aimed at redesigning the Internet architecture from scratch. This includes approaches such as flexible protocol stacks and information channels established on-demand, spanning over diverse physical network technologies [BJT+10]. This research direction is known as *revolutionary* or *clean-slate* meaning that the existing reference models should be fully redesigned, in order to enable efficient management of the communications' and services' infrastructure within an Internet environment. Within this community, it is believed that the redesign would allow for intrinsically embedding management hooks inside the protocols and the services, such that they become easily controllable in contrast to traditional solutions that emerged as add-ons. Indeed, a revolutionary approach would allow for implementing sophisticated resilient self-healing mechanisms, which can base their operation on collaboration between functional entities [Cha09]. On the contraty, the vision of an incremental, i.e. *evolutionary*, development of the Internet is also largely advocated within the research community [Cha08] [DDV+11] [WMR+11]. This

includes the improvement of existing protocols, including their self-healing features, as well as the refinement of existing management approaches and their extension into the device architectures [FFR+11] [BMQS08]. An advanced evolutionary approach from the past years is given by the introduction of Software Defined Networking (SDN) [HPD+15] [NHG+16] and Network Functions Virtualization (NFV) [NFV12]. Thereby, the devices are meant to offer programmable interfaces, which can be used to control network nodes and re-configure them on the fly - by so-called network apps developed in legacy programming languages, e.g. Python, C/C++ or Java. Combining this with the virtualization of the devices and the control on "virtual machine"/hypervisor level (the scope of NFV) allows for more flexible network control and management.

The ability of a network to employ self-healing mechanisms in order to preserve network functionality in case of failure occurrences or malfunctioning is of great significance. Especially, given the importance of the Internet, it is a major goal of this scientific work to lay the ground for the possibility for network operators to instrument a software based self-healing framework across the nodes of their network, and to utilize this framework, in order to increase reliability, robustness and QoS. In addition, the self-healing mechanisms of the framework (to be developed in this thesis) will enable the network to self-manage and reduce the involvement of the network operations personnel in some of the critical management tasks. The approach to self-healing, pursued in this thesis, belongs to the class of evolutionary approaches (refer to the elucidations in the above paragraph), since it does not advocate for a disruptive redesign of the network protocols and applications, but rather tries to implement additional components (i.e. a self-healing framework) that automatically manage and control the corresponding functional entities with respect to incidents.

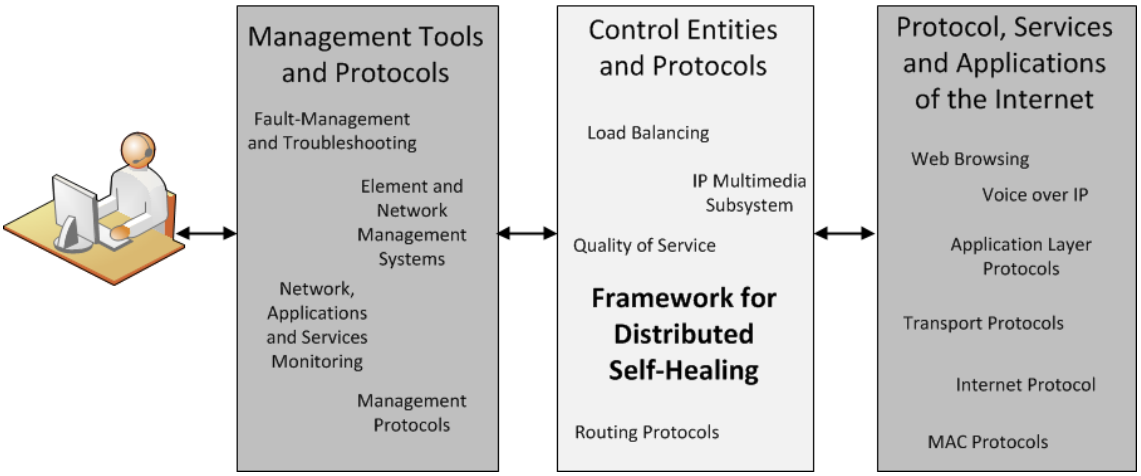

Figure 1.1: Identifying the Gap where the proposed Framework is required

In line with the above thoughts, the place of the envisioned *Framework for Distributed Self-Healing (FDH)* within the management and control hierarchy for communication based systems (e.g. Internet or telecom networks) is illustrated in Figure 1.1. FDH belongs to the set of control entities and protocols (also denoted as control plane according to [Gro]) that dynamically react to changing operational conditions by readjusting parameters of the corresponding protocols, applications and services. In that context, based on pre-supplied operational models, FDH strives to automatically resolve certain incidents occurring during the operation of a networked system. This is best realized by considering the knowledge and experience of the extremely skillful human experts managing and possessing deep know-how regarding the operation and maintenance of a particular network. Hence, there is a need to introduce and embed the knowledge of this

limited number of professionals into the managed network elements, such that a certain degree of automation is achieved with respect to network and service problems that can be solved by a set of distrubuted software components/agents.

The introduction of a framework with the FDH characteristics - as outlined above and positioned in Figure 1.1 - brings in further benefits regarding the economic site and the availability of end user services and applications. A clear advantage resulting from the adoption of the FDH mechanisms is the reduction of the human involvement, which gives potential for OPEX [2] savings in the network troubleshooting processes and tasks. Another benefit is that a framework such as FDH would increase the availability and reliability of the services (e.g. VoIP, video on-demand, ...) utilizing the network segments in question, because the automated resolution of (some) faults in the IP layer will inevitably reduce the down-time of the affected network infrastructure and will in turn affect positively the belonging MTTR (Mean Time To Repair) performance indicators. The main explanation for these considerations is that the network operators and end users would not have to wait anymore for human experts to put hands on the network and resolve the underlying challenges related to the faults that are removable by means of the emerging FDH framework (e.g. network problems of repetitive nature).

1.1.1 Problem Statement

Relevant survey results [AL10] declare that *"Most of the network management problems are repetitive in nature"*, which shows that there is a large potential for the application of automated self-healing control loop structures to resolve operational problems of a network/system during run time. The vision, with respect to the emerging framework for self-healing, is that the human expert's knowledge that is required to solve (for example) such repetitive problems for specific network/system in question, should be transferred into a self-healing (software based) framework running in the node/device architectures. This framework would provide means for automatic self-healing thereby performing Fault-Detection, Fault-Isolation, and Fault-Removal in an efficient manner. This implies that the involvement of the highly skilled network operations personnel should be shifted from being the key operational actor in the processes of Fault-Management and troubleshooting, to supplying knowledge to the FDH self-healing mechanisms in the network and maintaining, observing and improving the FDH control loop reactions in the long-term operation of the network.

Problem Statement: The problem, this dissertation deals with, is given by the challenge to

- identify the required FDH software components,

which operate in a distributed manner inside the network elements, and increase the self-healing capabilities of a fixed IP network infrastructure. Furthermore, the research of this thesis needs to prove the

- feasibility,
- efficiency and
- scalability

of the FDH framework for distributed self-healing.

[2] OPEX stands for OPerational EXpenditures

1.1.2 Research Hypothesis

Based on the above consideration, the following research hypothesis is formulated:

Research Hypothesis: The author of this work claims that it is possible to

1. design and specify a software agent based framework for distributed self-healing in fixed IP networks
2. select or devise efficient and scalable algorithms for the different tasks within the emerging self-healing framework
3. design the framework in a way that it optimizes its reactions and operational parameters (i.e. self-optimization features), in order to continuously improve the QoS of the network
4. design the framework in a way, such that the network operation personnel does not lose control over the network/system in question, but rather stays in the loop by monitoring and intervening (if required) in order to resolve problems or improve the operation of FDH
5. implement the proposed FDH framework for distributed self-healing
6. show that FDH can increase the availability of fixed IP networks
7. show that FDH can scale its operations with a growing size of relevant parameters (e.g. number of incidents)
8. show that FDH operates efficiently with minimal overhead (e.g. memory consumption and CPU utilization) within the network devices

The work presented from here on is meant to prove the claims of the above research hypothesis.

1.2 Objectives

The main objective of the current thesis is to derive the concepts, design principles and algorithms required to realize *scalable and efficient distributed self-healing with self-optimization features in fixed IP Networks*, i.e. the above discussed FDH framework. The challenge in designing an integrated architecture for self-healing (such as FDH) is to accommodate the variety of embedded resilient mechanisms of the existing network entities into an architectural framework targeting the goal to oversee and control the network/system in question, thereby reacting to erroneous states without interfering with the intrinsic resilient behaviors of existing protocols and services. Hence, the main objective of the current thesis translates into the need to develop such an architecture/framework by identifying the design principles and the major components that nodes/devices inside an FDH controlled network must be equipped with. This requires various components and interfaces, e.g. interfaces to monitoring entities, components facilitating the sharing of alarm/incident information and modules and methods for Fault-Isolation (i.e. root cause analysis). While striving to achieve the automated removal of faults (i.e. root causes for malfunctioning), the FDH components are naturally supposed to execute some Fault-Masking/Mitigation behaviors, in order to sustain basic network functionality and an acceptable network performance. At the same

time, the emerging framework should be equipped with a feedback mechanism that notifies/identifies on whether FDH can successfully cope with the challenges to the network/system, and should result in informing (i.e. escalating to) the network operation personnel in case of irresolvable issues.

1.2.1 Expected Scientific Output

The expected scientific output of this dissertation is the **identification of software components, which operate in a distributed manner inside the network nodes, and increase the self-healing capabilities of routers and end systems within a fixed IP network infrastructure**. The software components to be designed should be able to **proactively and reactively respond to faulty conditions**, i.e. on one hand failures should be predicted and avoided, and on the other hand, an automatic response to already existing faulty conditions should be realized. Correspondingly, algorithms to realize these processes are required. The expected scientific output includes **the evaluation of existing mechanisms, and where required, it is expected that new algorithms will be developed for the emerging framework**. For the selected or designed methods/algorithms/concepts, **extensive theoretical and experimental evaluation is expected to prove their effectiveness with respect to self-healing, self-optimization as well as scalability and overhead (e.g. CPU utilization or memory consumption) in the network elements**. Furthermore, it is expected **to experimentally show that the framework as a whole can function and successfully increase the QoS of a network/system**. For that purpose **a prototype should be implemented which should be consequently used to perform experiments based on various case studies of industrial and scientific relevance**.

As a result of this dissertation, **the scientific investigation of a series of concepts is provided, which have the potential to improve the reliability of significant parts of the Internet**. Moreover, the expected results promise that **traditional Fault-Management processes can be simplified for the network operations personnel, and correspondingly an OPEX (operational expenditure) reduction achieved**. Furthermore, the framework and belonging algorithms are supposed to be designed in a way that enables the realization of real-time *self-healing* behaviors whereby the reaction strategy is always optimized in a way that key performance indicators of the networked system in question are improved. That is, the **proposed architectural framework should incorporate *self-optimization* features**.

1.2.2 Validation Strategy

To evaluate the proposed concepts and mechanisms, a number of case studies are executed. Additionally, the scalability and overhead (e.g. memory consumption, CPU utilization, and response times) of the proposed framework are evaluated through extensive experiments with large data/models/networks. In order to realize these two aspects, a prototype of the FDH framework is implemented and deployed in a testbed, which was worked out in the scope of a large scale EU FP7 project. Subsequently, the case studies are executed, first only for a specific component/module/algorithm/process, and second for the framework as a whole as it operates in the above mentioned testbed. Thereby, the effectiveness of the framework and its single components is evaluated by measuring the network/system operation improvement, which is achieved through the FDH control/management activities for the network in question. Moreover, based on the case studies, key single components/modules/algorithms/processes are tested by pushing key input parameters (e.g. network size or input data size) to the extreme, in order to experimentally evaluate the scalability,

overhead and performance of the FDH mechanisms. In cases where appropriate and required, discrete event simulation and network simulation is used to verify certain properties of the designed FDH behavior/process/algorithm/component/module.

1.3 Structure of the Thesis

The current thesis is organized as follows: Chapter 1 is the present introductory chapter containing the motivation aspects and the objectives of the intended research activities. The following chapter 2 sets the context for the planned scientific work by laying down the basic terminology and reviewing a large amount of ongoing research/development activities and standards. The chapters 3 and 4 introduce the various views and understandings of Self-Healing and Self-Optimization, which are at the heart of this thesis. Thereby, a suitable definition of both terms is selected which is subsequently used in the up-coming chapters. Chapter 5 handles the aspects of research requirements analysis by defining the basic directions for the scientific work on the emerging framework for self-healing with self-optimization features. Based on the identified requirements, chapter 6 proceeds with the FDH framework specification by devising key node components and specifying the dynamic aspects (i.e. interaction flows) among nodes and components towards realizing the scalable and efficient self-healing with self-optimization features. Chapter 7 deals with the design and specification of the self-healing processes within the emerging framework thereby selecting and deriving innovative algorithms and interaction schemes towards achieving resilient behaviors. Similarly, chapter 8 defines a scalable and efficient framework for self-optimization with respect to the tentative actions from the various involved agents. Thereby, techniques from the domains of Machine Learning and Mathematical Optimization are utilized in order to improve the speed, overhead and effectiveness of the proposed algorithms. The following chapter 9 deals with the network operations personnel view on the emerging self-healing framework and elaborates on aspects such as the configuration of operational models and the escalation of faulty conditions - which are not resolvable by the self-healing mechanisms - to the network administrators. The following chapters 10 and 11 deal with the research prototype that was implemented in the scope of the current thesis and its integration into the testbed of a large scale European project. Based on these two chapters, the next chapters 12 and chapter 13 present a thorough scalability and overhead analysis of the proposed FDH framework, combined with a number of case studies that demonstrate the applicability of the derived components and illustrate the way self-healing with self-optimization is achieved, based on the presented research requirements and designs. Finally, chapter 14 concludes the thesis, relates the presented work to modern trends in networking and distributed systems, and outlines future research directions.

1.4 List of Publications

The following is the final list of publications that emerged in conjunction with the presented approach. These publications are referenced in the particular chapters across the thesis, for which a publication covers the core of the belonging content.

1. [TS14a] Tcholtchev, N.; Schieferdecker, I.: "Framework for distributed autonomic self-healing in fixed IPv6 networks". Int. J. Communication Systems, John Wiley and Sons Ltd, Volume 27, Issue 12, December 2014, Pages 4103-4125

2. [STP+15] Starschenko, A.; Tcholtchev, N.; Prakash, A.; Schieferdecker, I.; Chaparadza, R.; "Auto-configuration of OSPFv3 routing in fixed IPv6 networks," 2015 7th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Brno, 2015, pp. 196-205

3. [TS14b] Tcholtchev, N.; Schieferdecker, I.: "Framework for Ensuring Runtime Stability of Control Loops in Multi-agent Networked Environments", Transactions on Computational Science XXII, 2014, Volume 8360 of the series Lecture Notes in Computer Science, pp 64-92

4. [NTSR14] Nieminen, M.; Tcholtchev, N.; Schieferdecker, I.; Räty, T.: "Robust architecture for distributed intelligence in an IP-based mobile wide-area surveillance system", The Springer The Journal of Supercomputing, December 2014, Volume 70, Issue 3, pp 1120-1141

5. [WTVL13] Wodczak, M.; Tcholtchev, N.; Vidalenc, B.; Li, Y.: "Design and Evaluation of Techniques for Resilience and Survivability of the Routing Node", International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS), Volume 4, Issue 4, 2013, pages 36-63

6. [CWM+13] Chaparadza, R.; Wodczak, M.; Ben Meriem, T.; De Lutiis, P.; Tcholtchev, N.; Ciavaglia, L.; "Standardization of resilience and survivability, and autonomic fault- management, in evolving and future networks: An ongoing initiative recently launched in ETSI," 2013 9th International Conference on the Design of Reliable Communication Networks (DRCN), Budapest, 2013, pp. 331-341

7. [TP12] Tcholtchev, N.; Petre, R.: "Design, Implementation and Evaluation of a Framework for Adaptive Monitoring in IP Networks",EASe 2012, 9th IEEE International Conference and Workshops on the Engineering of Autonomic and Autonomous Systems

8. [TPS+12] Tcholtchev, N.; Cavalcante, A. B.; Chaparadza, R.:"Scalable Markov Chain based Algorithm for Fault-Isolation in Autonomic Networks" IEEE Global Telecommunications Conference GLOBECOM 2010 : 6-10 December 2010, Miami, Florida, USA. Piscataway, NJ: IEEE Service Center, 2010, 1-6

9. [TGV09] Tcholtchev, N.; Grajzer, M.; Vidalenc, B.:"Towards a Unified Architecture for Resilience, Survivability and Autonomic Fault-Management for Self-managing Networks", International Workshop on Service Monitoring, Adaptation and Beyond (MONA+), Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops : International Workshops, ICSOC/ServiceWave 2009, Stockholm, Sweden, November 23-27, 2009. Berlin [u.a.]: Springer, 2010, Lecture notes in computer science 6275 : Services Science, pages 335-344

10. [TCP09] Tcholtchev, N.; Chaparadza, R.; Prakash, A.: "Addressing Stability of Control-Loops in the Context of the GANA Architecture: Synchronization of Actions and Policies", 4th IFIP TC6 international workshop, IWSOS 2009, Zurich, Switzerland, December 9-11, 2009. Berlin [u.a.]: Springer, 2009. (Lecture notes in computer science 5918), pages 262-268

11. [TC10a] Tcholtchev, N.; Chaparadza, R.: "Autonomic Fault-Management and Resilience from the Perspective of the Network Operation Personnel", IEEE Workshop on Management of Emerging Networks and Services (MENS), IEEE Global Telecommunications Con-

ference GLOBECOM 2010 : 6-10 December 2010, Miami, Florida, USA. Piscataway, NJ: IEEE Service Center, 2010, pages 469-475 ISBN 978-1-4244-8863-6

12. [TC10b] Tcholtchev, N.; Chaparadza, R.: "On Self-healing based on collaborating End-systems, Access and Core Network Components" Access Networks, 5th International ICST Conference on Access Networks, AccessNets 2010 and First International Workshop on Automatic Networking and Self-Management in Access Networks, SELFMAGICNETS 2010. Springer, 2010, pages 283-298

13. [RTCM11] Rebahi, Y.; Tcholtchev, N.; Chaparadza, R.; Merekoulias, V.: "Addressing security issues in the autonomic Future Internet", 2011 IEEE Consumer Communications and Networking Conference (CCNC), 2011, pages 517 - 518

14. [KTP+10] Kastrinogiannis, T.; Tcholtchev, N.; Prakash, Arun; Chaparadza, R.; Kaldanis, V.; Coskun, H.; Papavassiliou, S.: "Addressing stability in Future Autonomic Networking", Mobile Networks and Management: Second International Conference ; MONAMI 2010, Santander, Spain, September 22 - 24, 2010; revised selected papers. Berlin ;Heidelberg [u.a.]: Springer, 2011, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 68, pages 50-61

15. [LZM+10] Liakopoulos, A.; Zafeiropoulos, A.; Marinos, C.; Grammatikou, M.; Tcholtchev, N.; Gouvas, P.: "Applying distributed Monitoring Techniques in Autonomic Networks", IEEE Workshop on Management of Emerging Networks and Services (MENS), 2010 IEEE Global Telecommunications Conference GLOBECOM : 6-10 December 2010, Miami, Florida, USA. Piscataway, NJ: IEEE Service Center, 2010, pages 498-503

16. [CTK10] Chaparadza, R.; Tcholtchev, N.; Kaldanis, V.: "How Autonomic Fault-Management can address current Challenges in Fault-Management faced in IT and Telecommunication Networks", AccessNets 2010 and First International Workshop on Autonomic Networking and Self-Management in Access Networks, SELFMAGICNETS 2010. Springer, 2010, pages 253-268

17. [CTS08] Chaparadza, R.; Tcholtchev, N.; Schieferdecker, I.: "Implementation of the UniFAFF Framework for Autonomic Fault-Management in ANA Networks", Proceedings of the Third International Conference on the Latest Advances in Networks : ICLAN'2008, Toulouse, France, December10-12, 2008, pages 75-81

Chapter 2

Setting the Context

This chapter sets the overall context and terminology, which are required to follow the presentation and contents of the research efforts elaborated in the current thesis. Initially, some basic terms are clarified and explained thereby preparing the background and foundation for the following discussions. Subsequently, a state-of-the-art analysis presents related research and development initiatives of the past years, in order to prepare the reader for the following conceptual and experimentation activities, as well as discussions and descriptions.

2.1 Background

Right at the beginning, some basic concepts should be clarified that are required for developing the theoretical aspects of the work presented in the current thesis. This is imperative because there seems to be different perceptions on some of the key terms related to Fault-Management, Resilience, Survivability and self-healing for networked systems.

Within this thesis, Fault-Management is similarly understood and handled as in the FCAPS approach, which is part of the TMN standard [ITU00]. That is, Fault-Management is perceived to consist of, and to be dealing with, the following functions: Fault-Detection, Fault-Diagnosis (also referred as Fault-Localization or Fault-Isolation), and Fault-Removal[1].

Dealing with incidents (faults/errors/failures) and alarms is the main objective of Fault-Management. The semantics of these terms lead again to substantial confusion [Wal09], as within different domains and standards there seems to be various understandings, or at least multiple perceptions relating to these key aspects. Hence, a simple and clear terminology needs to be adopted that can serve as the base for introducing self-healing with self-optimization features as envisioned by FDH. In this line of thought, the relations between faults/errors/failures and alarms need to be clearly defined and established. There exist two stable and consistent models (as elaborated in [Wer07]), which are to be taken into account - the Failure-Fault-model of Kopetz [Kop82] and the model of Laprie [LAK92] [Lap92]. The Failure-Fault-model of Kopetz [Kop82] defines a Fault and a Failure as being the event of malfunctioning (*Fault*) and the state of mal-

[1]To be precise: [ITU00] states following function groups as the core parts of Fault-Management. (1) RAS (Reliability, Availability and Survivability) Quality Assurance, (2) Alarm Surveillance, (3) Fault Localization, (4) Fault Correction, (5) Testing, and (6) Trouble Administration. In order to establish a straight link to self-healing, the functions above are simplified, the most important ones are picked, and the overall process is presented as consisting of Fault-Detection, Fault-Diagnosis, and Fault-Removal.

functioning (*Failure*) of a system. The model of Laprie [LAK92] [Lap92] is a more complex one and provides additional structures by involving the concepts of *fault*, *error*, and *failure*. Thereby, the *fault* is seen as the root cause of the malfunctioning, the *error* as the state of malfunctioning, and the *failure* stands for the fact that certain components fail to deliver their services, which is observable at the belonging interfaces. Hence, aligned to the sophisticated model of Laprie, this thesis adopts an approach to the fault/error/failure/alarm/incident-understanding as defined by [ITU00][ALRL04] and referred by [SS04a] [Ran00]. The adopted approach can be summarized as follows:

- *fault* - a root cause for the malfunctioning of a system/network. That is, it constitutes a structural, configurational or bahvioural cause for an erroneous state. Faults can further propagate as one or more *errors*.
- *error* - a behavioral or computational deviation from correct pre-specified behaviors, values or conditions. An error is either a direct consequence of a fault or of another error, and propagates as a *failure* or as an additional error.
- *failure* - an error, which is observable at service/interface level. That is, a failure is a deviation between the expected result at a service/interface and the delivered/obtained result.
- *incident* - in the following, faults/errors/failures are often referred by the term *incident*.
- *alarm* - Alarms are understood as warning notifications concerning detected deviations from normal operational states and conditions. An *alarm* may or may not represent a fault/error/failure. For example, an alarm might be generated in case a particular threshold is being approached, which would constitute a proactive warning for a future fault activation - however the fault has not occurred at the time of alarm generation.

Coming back to the above discussion that Fault-Management consists of Fault-Detection, Fault-Isolation, and Fault-Removal: Fault-Management can be easily explained by relating its processes to the above definition of faults/errors/failures/alarms.

- *Fault-Detection* - stands for the process of detecting that a fault has been activated. In that sense, normally a failure would be detected by a monitoring component, which would indicate that a fault has occurred within the network/system in question. It is also possible that an alarm is generated and observed as a result of fault occurrences within the corresponding network/system.
- *Fault-Isolation* - stands for the process which is responsible for correlating the amount of detected incidents and generated alarms, and localizing the fault (root cause) for the experienced erroneous state (detected within the Fault-Detection process).
- *Fault-Removal* - stands for the process of removing and resolving the faults, which are isolated in the course of the Fault-Isolation task.

The combination of the above processes yields the *automated self-healing in the long-term operation of the network/system*, which is one of the key functions of the emerging FDH framework as elaborated in the coming chapters. This concept is analog to the concept of Autonomic Fault-Management as presented in [Cha09] - i.e. the automated detection of faults, followed by the

automated Fault-Isolation and Fault-Removal. Furthermore, [Cha09] gives some additional ideas that have been considered to some extend within the emerging FDH design, especially the statements in the following quote saying that Autonomic Fault-Management involves *"cooperative and collaborative mechanisms implemented in nodes and the network as a whole, to automatically detect incidents, share knowledge about incidents, diagnose/localize the root cause of the observed incidents, and finally remove the fault (root cause), throughout the operation lifetime of a node and the network as a whole"* [Cha09].

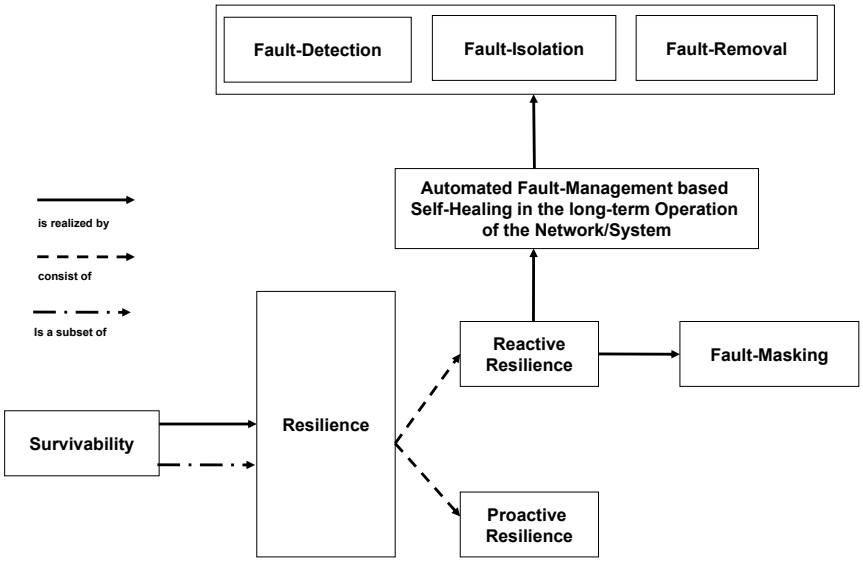

Figure 2.1: Relations between Fault-Management, Resilience, and Survivability

The concepts and terminology presented hitherto are in direct connection to the basic network functions of Resilience and Survivability. The relations and interplay among the overall set of notions and concepts is illustrated in Figure 2.1. Relating to Figure 2.1, Resilience is perceived as the ability of the network to automatically recover from some undesirable state (e.g. malfunctioning), or different faulty conditions/states, and to *"provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation"* [Res17]. The concept of Resilience can be additionally zoomed in and structured into Reactive and Proactive Resilience. A network/system providing Reactive Resilience has the capability to react to the presence of a fault by employing some Fault-Masking and/or remediation techniques [RMD05]. Obviously, given the above understanding, the self-healing based on automated Fault-Management[2] - including Fault-Detection, Fault-Isolation and Fault-Removal - can be seen as a means to achieving Reactive Resilience in the long-term operation of the network/system. To complement the picture on Resilience in Figure 2.1: a component/protocol/service/network/system with Proactive Resilience features tries to predict and adapt to future fault activations in the network/system [RMD05].

Focusing further on Survivability in Figure 2.1: Survivability is defined as *"the ability of a network to recover the traffic in the event of failure, causing few or no consequences for the user"* [VPD04]. Survivability is also understood as *"the capability of a system to fulfill its mission, in*

[2]This type of self-healing is also sometimes referred to as Autonomic Fault-Management.

a timely manner, in the presence of attacks, failures, or accidents" [FLL+97]. As described in Figure 2.1, Survivability is as a subset of Resilience. Especially, it stands out as the aspect of Resilience that tries to sustain the performance of the network in the presence of failures and to hide the disruptions below the service level from the user. To summarize: various concepts such as Fault, *error*, *failure*, *incident* and *alarm* were laid down, being the notions which build the base for the functions and processes of Resilience, Survivability and Fault-Management. Based on these understandings, the FDH framework for distributed self-healing with self-optimization features is specified and designed in the coming chapters.

2.2 Related Work

Having discussed and clarified the basic concepts and terms that relate to the emerging FDH framework, the following sections represent a state-of-the-art review of related work and adjacent areas to the self-healing aspects, which are at the heart of this thesis. The first topic that is handled deals with the overall network resilience and related standards and research efforts. Subsequently, an insight in network disaster recovery is given, i.e. into the techniques as well as implemented mechanisms and processes for reestablishing a network after a major disaster (e.g. earthquake, flooding, terror act ...). Afterwards, the area of Network Management is looked into thereby elaborating on legacy standards and practices from this vital topic for modern and future networks. The topic of Autonomic Computing and Networking is the next one to be handled in a row. Autonomic Computing/Networking was one of the key topics and large hopes of the past decades regarding the challenge of stabilizing networks and computer based systems (in general) by increasing their reliability and reducing the belonging management overhead (through the introduction of adaptive self-X type of behaviors within the corresponding software components). The following reviews of Control Theory approaches and Policy based Network Management techniques are in close relation to the topic of autonomics and self-adaptations for networks and systems. The last three sections focus on the topics of Dependable Systems, Agent Systems, and Software Defined Networking and Network Functions Visualization. These three topics are closely related to the emerging FDH framework as they incorporate different reliability aspects as well as the notion of entities/components that can exercise network control functions and can improve the operations of a network/system.

2.2.1 Network Resilience

Research in the area of Network Resilience has a long history in the world of telecommunications, networking and distributed systems. In general, Network Resilience is targeted as a property that is integrated in the network components/entities. There are various approaches to implementing Resilience in current networks including mechanisms for protection and restoration, e.g. (G)MPLS [Man13], SONET/SDH [TEL09], FDDI [AJ94], and Token Ring [LS98]. Further mechanisms can be intrinsically implemented in the protocol modules, for example as a reaction to ICMP [Pos81] notification of errors along a path (e.g. in the case of PMTU [MD90] discovery), or link/node failure detection and automated adaptation of routing and the FIB (Forwarding Information Base) of involved routers in the case of OSPF [Moy98]. The Resilience of intra- und inter-domain Routing was reviewed in [LRS+05] [RMD05] with respect to different known issues, thereby discussing on typical reactive and proactive approaches.

As one can observe, Network Resilience can be implemented on various layers within the proto-

col stack, such as Layer-2 in the case of FDDI and Token Ring, Layer-2.5 in the case of MPLS, Layer-3 in the case of ICMP and its error codes, and L4 in the case of TCP resilient mechanisms and reaction to errors (e.g. PMTU and CWND[3] adaptation schemes). Research such as [TH01] and [SS04c] handle the concept of Multi-layer Resilience in telecommunications including IP networks, which is a key topic to be taken into account when constructing a framework for distributed self-healing such as FDH.

On the level of public entities, various resilience aspects are addressed by European and national agencies like ENISA [ENI17] and BSI [BSI17]. Thereby, especially the aspects of cyber-security as well as protection of information and data privacy are thoroughly addressed. Furthermore, the resilience and security of critical infrastructures (energy and mobility networks, telecom and public networks ...) are subjects of high relevance for such public entities and their related activities, including workshops, security indicators, certification processes and assessments, technical recommendations and protection profiles.

Focusing on some recent research activities, it can be observed that Network Resilience is substantially prepared during the phase of network design. This includes on one hand the topology of the network nodes, and on the other hand the choice of protocols, algorithms and restoration schemes for the network system in question. [TDM16] distinguishes between *design for robustness* and *design for resilience*. *Design for robustness* is understood as the approach to construct a system/network with the capability to sustain unexpected challenges and perturbations thereby mainly relying on redundancy and overdimensioning, in order to facilitate the handling of critical situations. *Design for resilience* stands for extending the capabilities of the involved components such that they can adapt to erroneous states and can keep up a certain level of performance. Eventhough this understanding differs from the wide perception hitherto, the authors of [TDM16] acknowledge, that resilience is also widely understood as the adaptation and survivability capabilities of single or multiple components as well as the overall design and (over)dimensioning of networks/system, in order to prepare them for challenging conditions. Based on the introduced distinguishment between *robustness* and *resilience*, the authors of [TDM16] elucidate on both aspects in the context of systems-of-systems (SoS), which is perceived as *"large-scale integrated systems which are heterogeneous and independently operable on their own, but are networked together for a common goal"* according to [Jam08]. Thereby, the goal is to provide a framework that allows to estimate the cost of both approaches for an SoS design undertaking. The framework comprises a formal model and a set of metrics that allow to judge on which approach is more suitable for a particular SoS. The applicability of the proposed framework is shown on a case study utilizing simulations relating to command-and-control in UAV missions and scale free networks. Moreover, [MMMS16] deals with the topic of realizing network resilience via self-healing within smart grid environments. Thereby self-healing refers to the recovery of network connectivity over redundant links in case of link/node/infrastructure failures. Therefore, a theoretical framework is devised that allows to calculate the ratio of served nodes given a certain level of network damage. Hence, the proposed approach constitutes a mix between network planning - in terms of backup links/paths - and distributed routing protocols, such as OSPF, RIP, and IS-IS. Indeed, the research in [MMMS16] supports the network planning in the sense of infrastructure and software components, in order to increase the resilience of a smart grid environment. Another exciting research is given by the elucidations in [PRS16]. [PRS16] approaches the topic of network resilience on the level of efficiently configuring traditional link-state routing protocols (such as OSPF and IS-IS), in order to establish a resilient IP-network layer routing topology. The resulting mathematical problem is based on different resiliency related constraints - e.g. various routing protocol parameters

[3] TCP Congestion Window

such as link weight or routing area size - and turns out to be NP-hard, incorporating various graph based artifacts and belonging matrix representations. Correspondingly, a multi- objective optimization procedure is applied, in order to automatically generate link-state routing configurations that increase the network resilience in turn. The automatically obtained configurations aim at realizing a robust network service (i.e. packet routing/forwarding) in the face of link and node failures, thereby reducing and even avoiding traffic congestion after rerouting has been accomplished.

Network Resilience with respect to Information-Centric-Networking (ICN) is at the core of [RPNR17]. ICN is a paradigm that was researched in the past years and constitutes a view on the network infrastructure as a vehicle for transmitting information and not just packets. Hence, the focus is set on indexing, tagging (e.g. by assigning IP addresses) and naming pieces/chunks of information and not only network hosts, as in the case of traditional host-to-host communication oriented networking. In this line of thought, resilience is understood not only as enabling the network nodes to forward and deliver packets, but as facilitating the network as a whole to meet user demands with respect to various types of information. In that scope, [RPNR17] reviews multiple reasons for network impairments (e.g. human errors, cascading failures, large scale disasters ...) and elucidates on different methods for increasing the resilience of communication oriented networks, as well as describes the relation between these methods with the goal to provide robust network services. Furthermore, ICN is put in close relation to P2P-networking and fundamental *named data routing* principles are presented and discussed, thereby putting emphasis on data routing on top of overlay networks (such as the widely researched P2P models). Finally, the proposed resilience approach with respect to *named data* and *overlay* routing is described: this approach to resilience is based on SDN[4] and is a modern concept for network control and management, which is presented later on in section 2.2.9. Network resilience in conjunction with an SDN based control plane is also the main topic in [BDRG17]. The SDN control plane is an extra substrate, which is separate from the data plane - solely responsible for simple traffic forwarding - in the scope of SDN[5]. Thereby, the control plane contains special components (SDN controllers) that steer a number of network elements with respect to a variety of networking functions, such as routing, QoS, security, mobility etc. Given the fact that SDN is of paramount importance for trends such as 5G, Mobile Broadband, (Mission-critical) IoT/M2M and further, the network planning regarding the SDN controller components is of key relevance for future resilient networks. Thereby, the placement of the controllers - i.e. which controllers are responsible for which nodes and functions - is paramount for increasing the resilience of the production network, also denoted as data plane in SDN. The SDN controller placement plays a vital role when it comes to aspects such as latency for the communication between network elements and controllers, reliability, scalability, and overall network performance. [BDRG17] formally defines the controller placement problem (CPP) in relation to a set of metrics specified within the same work. Subsequently, [BDRG17] applies optimization algorithms for the solution of CPP with the goal of identifying solutions which can provide SDN controller placements with maximal reliability and resilience awareness. Another modern resilience solution based on SDN and NFV[6] is given by the ANSWer[7] framework for combining NFV and SDN for Network Resilience strategies [MGSF16]. Thereby, a feedback (autonomic) control loop is realized through the utilization of network virtualization and SDN controllers that steer the virtualized network components. In that scope, ANSWer implements different monitoring procedures that

[4] SDN stands for Software Defined Networking

[5] More details on SDN are given in section 2.2.9.

[6] NFV stands for Network Functions Virtualization and is described in conjunction with SDN in 2.2.9

[7] ANSWer stands for *Architecture for combined Network functions virtualization and SoftWarE-defined netwoRking features*

gather statistics for numerical distributions with respect to relevant key performance indicators. In case a particular threshold - with respect to the expected value and standard deviation of the KPI distribution - is exceeded, ANSWer undertakes different mitigation actions which are realizable based on SDN and NFV principles, e.g. different security/IDS/IPS/firewall types of actions to handle a symptomatic erroneous state. Corresponding actions and strategies are elaborated in detail in [MGSF16]. As a final exciting research reviewed in the selected sample: the resilience of enterprise networks is the main focus of [Sam16]. Thereby, key resilience relevant parameters of an enterprise network are steered and optimized based on corresponding utility functions and algorithms utilizing biological principles as observed in the scope of aunt colony behaviors. The main optimization activity relates to the dynamic reorganization of the enterprise network infrastructures in logical clusters, which are further connected over a backbone network, with the goal to improve the overall network resilience and influence aspects such as traffic congestion and latency. Having reviewed various research efforts relating to Network Resilience, the following section moves on with the topic of Network Disaster Recovery dealing with the reestablishment of a network infrastructure after a catastrophic event.

2.2.2 Network Disaster Recovery

The topic of Network Disaster Recovery deals with methods and processes for restoring network infrastructures following catastrophic events, such as terror attacks, earthquakes, flooding, war situations etc. In such cases, the physical infrastructure is normally damaged to an extent that requires the network operator to undertake immediate actions as to enroll some temporary resilient solutions and connect the catastrophic area to nearest access to the global Internet/telecommunication network(s). The connection establishment is extremely important in order to restore communication capabilities and facilitate the emergency response teams in their vital work. Publications such as [AMBK+07] [Mor11] [OWM11] [Ran11] describe the network operator's view and established processes with respect to Network Disaster Recovery. Thereby, [AMBK+07] describes a 911-network on wheels solutions (911-NOW) that is intended to restore network connectivity through wireless communication after a catastrophic event. Thereby, the network on wheels incorporates different types of technology, such as base station routers (BSR) not requiring any pre-existing infrastructure combined with the deployment of various technologies on different ISO/OSI layers, such as Mobile IP, Proxy Mobile IP, WiFi, WiMAX, and potentially Software Defined Radio (SDR), with the goal to provide capacity and coverage on-demand. Thereby, the BSRs and the attached infrastructure are intended to have auto-configuration capabilities for aspects such as addressing, routing and mobility. Furthermore, key challenges such as the network management for a "network on wheels" providing a single-cell ad-hoc solution is elaborated, focusing on the auto-configuration as a vital issue. Furthermore, [Mor11] describes the network emergency management procedures of AT&T[8] aiming at fast and efficient Network Disaster Recovery. Indeed, the AT&T implements a so-called network emergency management and business continuity program that defines and supports the implementation of the network recovery procedures. This includes sophisticated strategies realized during the network planning phase including network architectures that support diverse network paths and automated rerouting for fast reactions to cable cuts or node failures. In case of entire central office (or Network Operation Center) damages, special trailers are instrumented which are operated by full-time and volunteer employes of AT&T. Indeed, the AT&T Network Disaster Recovery fleet consists of various types of vehicles such as portable COLTs (Cell On Light Truck), special vehicles for reestablishing network telephony functions

[8] AT&T is a large US network provider.

such as PBX, or more comprehensive COW (Cell On Wheels) trucks. A remarkable deployment of the AT&T fleet is given by the network damage response after the September 2001 World Trade Center Disaster in New York, where a complete Network Operation Center was destroyed in one of the WTC towers. The reaction processes are described in [Mor11] providing valuable insights with respect to real world large scale Network Disaster Recovery. In addition, the emergency recovery plans and procedures of Verizon[9] are the topic of [OWM11]. [OWM11] provides fundamental criteria for an efficient Network Disaster Recovery plan upon which the currently implemented Verizon approach has been realized. This includes the definition of relevant KPIs with respect to different areas such as financial implications as well as stakeholder and customer impact. The general plan involves continuous management processes such as risk identification, risk mitigation, risk elimination, business impact analysis and strategy and plan development/adaptation. All these aspects are accommodated in a specially defined Emergency Operations Center (EOC) that can be viewed as a separate function or functional unit within a network provider. It is also possible that particular Network Operation Centers take over the role of an EOC for a corresponding damaged area. The EOC is then responsible for coordinating the various mobile equipment carrying vehicles (such as trailers, COLT, COW ...) towards the reestablishment of network connectivity and services. Further items, which require to be managed (tracked) are given by the need for efficient information distribution among the involved teams like expenditures and statistics related to the recovery procedures. That implies the need for data management tools but also for real time communication facilities such as conferencing systems etc. Moreover, [OWM11] elaborates on the role of governmental institutions and the required collaborations between network operators and public entities in the course of network recovery after a catastrophic event. In line with these valuable reports, [Ran11] analyzes the efforts and actions undertaken by Chinese telecom providers and authorities after the devastating Wenchuan earthquake in 2008. Thereby, the priority was initially set on restoring PSTN services in conjunction with the provisioning of wireless communication links, such that rescue teams can approach and coordinate over the large and severely devastated earthquake area. The first challenge was to ensure the timely power and fuel supply for the network and to remediate the network congestion resulting from traffic rerouting after the experienced network damage. Afterwards, a three phase stage process was implemented: the first stage being the restoration of communication services from the disaster areas to the outside world, the second second being the reestablishing the communication on county level, and the third stage being the complete restoration of the communication within the catastrophic area and the initiation of network rebuilding processes. Thereby, the emergency equipment was delivered by army helicopters, sometimes brought on vehicles or even by foot in order to establish communications at the foremost edge of rescue operations. A couple of research areas are identified - as of 2011 based on the Wenchuan experiences - and are related to aspects such as information exchange, information processing, Software Defined Radio, sensor data, Machine-to-Machine communication and further, which were addressed in the past years.

Looking at general scientific activities regarding network disaster recovery: various methods for design and deployment of networks and services in emergency areas have been the topic of research during the past years. [MSW16] deals with the aspect of requirements management and selection for the design of a recovery capable networks and classes of applications/services.This includes the description of a process for requirements handling and querying as well as the belonging user interface to enable the requirements identification for Network Disaster Recovery, based on the specifics of the particular network in question. In addition, [PLG+16] deals with the process of virtual network embedding for efficient recovery of networks and services after a large scale dis-

[9] Verizon is another large scale US network service provider.

aster. Since many applications and services are migrating to virtualized cloud infrastructure, there is a need for smart deployment process for the belonging network structures and virtualized components, such that a successful restoration strategy can be implemented. The intelligent placing of virtualized components/links/services/apps and resulting recovery schemes are summarized under the term *Progressive Recovery* [PLG+16]. [PLG+16] proposes belonging algorithms for data centers and networks in general and provides a proof of concept based on OPNET++ simulations. A closely related approach is presented in [HGK16] where a method for deploying an emergency service on a disaster recovery site is presented. This deployment is again guided by principles of virtualization and auto-configuration, which are captured in a corresponding process. The goal is to enable a resilient deployment of vital infrastructure for a potential catastrophic area.

Another interesting approach to Network Disaster Recovery is given by the idea to use mobile phones in the catastrophic area for the purpose of establishing connectivity to a certain network access point. Examples for research activities in this area are given by [PMT16] [LCP16] and [MSBY16]. [PMT16] presents a model for establishing ad-hoc wireless networks (using on-site mobile devices) and routing optimizations, which builds on a spanning tree multi-path routing topology and load balancing, in order to avoid wireless link overloads whilst maximizing the chance of traffic to successfully leave the catastrophic area. The proposed model leads to a linear optimization problem which is correspondingly solved to achieve optimal routing configuration. The model is validated and its feasibility shown in the scope of Matlab simulations. Moreover, [LCP16] investigates on how different mobile devices can be automatically connected in order to span the gap between a disaster area and a telecommunication network. For that reason, a system called TeamPhone is proposed and prototyped consisting of two main components: 1) a messaging systems taking care of basic connectivity in terms of ad-hoc networking thereby enabling the communication across rescue teams, and 2) an emergency system called *self-rescue* that enables the communication between rescue workers and trapped survivors that need to be dug out and/or aided. In that line of thought, an architecture for on-site auto-configuration of mobile devices in the scope of network disaster recovery is proposed in [MSBY16]. It includes various abstractions of stratums (i.e. communication layers) for the on-the-fly utilization of mobile devices and wireless visualization techniques to establish required virtual wireless networks towards a network access point. The key abstractions are referred to as WMCA (Wireless Multihop Communication Abstraction) and TDRAN (Tree-based Disaster Recovery Access Network). The main goals of the proposed architecture are given by the need to efficiently reestablish Internet connectivity, the utilization of commodity mobile devices, and the need for easy to apply mechanisms to extend an existing wireless network. Corresponding existing protocols (e.g. for auto-configuration or routing) are reviewed and mapped to the layers and artifacts of the proposed framework. Furthermore, various required interaction flows are captured in the form of sequence diagrams towards the provisioning of ad-hoc wireless communication infrastructure for rescue teams and survivors.

2.2.3 Network Management

Traditionally, network management deals with various configuration and monitoring aspects of a network node/element as well as the network as a whole. Thereby, the management of specific (e.g. coming from one vendor) network nodes/elements/devices is handled by so-called EMS (Element Management Systems), whilst the management of the overall network is dealt with by Network Management Systems (NMS) overseeing the overall infrastructure and normally integrating the underlying EMS instances. In addition, different services and business aspects of a telecommunication infrastructure are considered and managed on top of the EMS and NMS level. Thereby,

the pure management systems are often denoted as Operation Support Systems (OSS) whilst the business related activities are handled within the scope of so-called Business Management Systems (BSS). These concepts are mainly introduced and discussed within various standardization and international fora, such as the Telecommunication Management Forum (TMF) [TMF17] with its eTOM Bussines Process Framework [eTO17] and ITU-T [ITU17] with the belonging Telecommunication Management Network (TMN) standard [ITU00]. Thereby, as previously mentioned, the TMN standard specifies the main goals of network management as handling the processes of Fault-, Configuration-, Accounting-, Performance-, and Security-Management - some of which are further discussed and developed across the current thesis. In addition, the ISO20000 based ITIL [ISO11] standard for the management of IT systems should be mentioned. It defines a variety of procedures and reports on best practices relating to Service Design, Service Operation, Change Management and Incident Management, to give some examples. Typical protocols implementing different aspects of the above discussed management architectures are given by SNMP [Sta98], Remote Network MONitoring (RMON) [Sta98], Common Management Information Protocol (CMIP) [Bla94], and the Network Configuration Protocol (NETCONF) [EBBS11], to name some of the most prominent examples. Furthermore, the topic of network monitoring plays an important role in enabling the implementation of the variety of management architectures and approaches. Prominent protocols are given by SNMP and belonging MIBs [Sta98], NetFlow [Cla04], IPFix [Cla08], and the PCAP (Packet Capture library) [PCA17].

Beyond the above listed traditional network management standards and protocols, the concept of In-Network Management has been developed in the scope of different European projects, such as FP7 4WARD [4WA17], FP7 AutoI [Aut17a] and FP7 UniverSelf [Uni17]. Thereby, the idea of In-Network Management is closely related to the topic of Autonomic Computing and Autonomic Network Management as reviewed in section 2.2.4. The basic idea is to enable the integration of different management components, mechanisms, and protocols, including management modules (e.g. agents) inside the devices. Based on this integration, it is intended to provide an in-built management functionality across multiple levels that extends into the OSS/BSS architectures, thereby increasing the level of self-management and reducing OPEX, whilst meeting pre-defined Service Level Agreements at the same time. Thus, autonomic concepts such as feedback control loops can be utilized and various aspects like self-healing and self-configuration (e.g. Plug-and-Play for network devices) realized in the scope of In-Network Management. All these considerations have close relation to research domains such as distributed network management, as reviewed in the coming paragraphs. In addition, architectures and design concepts like 4D [GHM+05], *"Knowledge Plane for the Internet"* [CPRW03] and CONMan (Complexity Oblivious Network Management) [BF06] aim at defining design principles for network protocols and components that increase the level of manageability and control for distributed architectures. Thereby, these approaches define different information sharing principles (e.g. Dissemination Plane [GHM+05] or Knowledge Plane [CPRW03]) and various decision points where intelligent and integrated system management is to be implemented - for instance based on cognitive systems [CPRW03].

Recent development in network management very often move towards accelerating and simplifying involved aspects and processes. [CLCG15] describes a network management system implementing the concept of auto-configuration for various heterogeneous devices. This includes the identification of new devices once brought into the system as well as the determination of management aspects for these devices, which are to be handled by the belonging network/systems management infrastructure. These processes encompass tasks such as the discovery of relevant characteristics of the network nodes (e.g. processor type, deployed software, NIC types ...) and the subsequent download of relevant management policies based on the discovered characteris-

tics. [LCCG17] constitutes a similar approach for the management of various types and classes of network and application/service topologies, including traditional TCP/IP networks as well as domain specific networks (e.g. field buses such as SCADA, CAN, ...). Thereby, the devices in question have the possibility to be part of an auto-discovery procedure which is in turn concluded by the download and application of required management policies to special network management nodes, from which the network is monitored and steered.

The usage of mobile agents[10] to perform network management tasks is one of the traditional research topics in the area of network management with [BPW98], [KX02] and [DLC03] providing typical examples. [BPW98] discusses the application areas of mobile agents for network management. Thereby, special properties of software agents are pointed out and a so-called navigation model is proposed that determines the mobile agents' behavior whilst conducting its assigned tasks. The usage of mobile agents is elucidated in the light of the FCAPS framework and the application to each of the frameworks' domains correspondingly discussed. Thereby, different technologies are considered for tackling tasks related to Fault-Management, Configuration-Management etc. For instance, the usage of SNMP [Sta98] and Corba/IDL. [KX02] constitutes another classical architecture utilizing mobile agents with the goal to increase the scalability of network management processes, mainly SNMP based centralized approaches. Hence, the so-called MAN - Mobile Agent Network Management framework - is devised. The MAN combines SNMP with a distributed mobile agent based approach. The main idea is as follows: instead of periodically querying a node from a centralized location, mobile agents are prepared/pre-configured and sent to the node in question, in order to locally gather and analyze information. Thereby, they interact with the local SNMP agent and can implement different filters, analysis algorithms or even re-configuration schemes for the local device. Subsequently, the results from the local activities (e.g. monitoring information) are communicated back to the Network Management System and made available to the network operations personnel. [DLC03] provides an overview of the usage of mobile agents for distributed network management, again with goal to remediate the scalability issues of centralized network management with the growing number of network segments to be managed. The mobile agents approach is meant to: 1) conserve network bandwidth by 2) reducing network management traffic and 3) increase management efficiency by bringing the management activities as close as possible to the devices whilst implementing aspects like data aggregation or network analysis within the local mobile agents. The utilized technologies are given by SNMP and RMON [Sta98], whereby the SNMP MIBs are grouped and hierarchically analyzed for an improved handling in the scope of distributed network management. Another version of distributed network management is given by the utilization of P2P infrastructures as described in [BTadG+05]. Thereby, DHTs (Distributed Hash Tables) are used in a P2P-overlay in order to store management information and enable the indexing and localization of pre-configured network management agents. DHTs are self-organizing overlays spanning over the physical network and in that case facilitate the coordination between agents, in order to perform different monitoring tasks, e.g. related to Fault- or Performance-Management. Again, the goal is to overcome the limitations of centralized network management by providing a flexible machinery addressing various tasks (especially monitoring) for the FCAPS areas. Examples of monitoring agents and tasks are given by the gathering of NIC-status information, DHCP-status information, connection status, DNS configurations etc. Finally, [KF13] represents a new wave of approaches to network management, which utilize aspects from the area of Software Defined Networking (elaborated in detail later in this chapter). Thereby, [KF13] elaborates on the challenges to traditional network management and argues that the vendor specific configurations of devices and fast changes of

[10] A detailed classification of the various types of mobile agents is given in section 2.2.8

networking conditions are among the largest challenges. Hence, the SDN concept could be used to quickly react to network condition changes whilst at the same time provide a unified interface for management policies which are to be supplied by the network operations personnel. Indeed, [KF13] elucidates on the role of different layers and planes in the scope of network management and proposes a high-level policy language to utilize in that context. Having presented different research and development activities in the area of network management, the next section proceeds with the closely related topics of Autonomic Computing and Networking.

2.2.4 Autonomic Computing and Networking

The vision of Autonomic Computing and Networking has been strongly pursued in the first decade of the 21st century. Indeed, a number of research activities including large scale scientific programs were launched as to improve the self-manageability of emerging systems and networks through the introduction of autonomic entities and protocols implementing feedback control loops. Thereby, the IBM white paper [ibm05] on autonomics can be seen as one of the starting points for research and development in the are of Autonomic Computing and Networking. [ibm05] defines a four self-* features which should be implemented by an autonomic system, i.e. self-healing, self-optimization, self-configuration and self-protection (relating to security aspects). Indeed, the aspects of self-healing and self-optimization are at the heart of the current thesis and are thoroughly reviewed in the coming chapter 3 on self-healing and chapter 4 on self-optimization. In order to achieve these properties of an autonomic system, feedback control loops are proposed as the main ingredient to implementing behaviors regulating a variety of system/network operational aspects. Thereby, [ibm05] proposes a generic feedback control loop model for autonomic systems, which is presented in Figure 2.2. The model basically describes the phases of a control loop implemented by an Autonomic Manager that steers the operations of a Managed Element (e.g. a network protocol). To achieve this, the Autonomic Manager uses so-called *Sensors* in order to monitor the Managed Element. Subsequently, the monitored data is analyzed and a plan is obtained as to how to adjust relevant parameters of the Managed Resources. Finally, the planned actions are executed on the Managed Element over an *Effectors* interface, in order to complete the feedback control loop structure. Thereby, different types of *Knowledge* are at the heart of the proposed control structure in Figure 2.2. This might include various operational artifacts such as Fault-Propagation Models, Causality Models, Dependability Models and others, which are handled in the scope of the current thesis. Further aspects of the IBM MAPE control loop are discussed on various occasions across the thesis.

Based on the above presented initial considerations of [ibm05], a number of conceptual works [SAL06] [DDF+06] [KC03] [ea06] [CAS16] [DAS09] [BJT+10] discussed and derived approaches to achieving autonomic network and systems' management. In general, these works define the self-* features which they perceive an autonomic system should provide (i.e. self-optimization, self-healing ...) and characterize those. Furthermore, the corresponding control loop phases are presented and defined in a specific manner thereby mapping different technologies to the feedback control steps - e.g. policies [SAL06] [DAS09] or executable plans [ea06] [CAS16]. Moreover, various layers of control loop interactions are often specified as in the case of [SAL06]. In addition, approaches such as the Autonomic Network Architecture [BJT+10] aim at re-designing and extending traditional protocols and network infrastructures with intrinsic interfaces and hooks (e.g. sensor and effector interfaces - see Figure 2.2) as to facilitate increased manageability and correspondingly the implementation of autonomic behaviors. Many of these research and development concepts emerged in the scope of relevant fora and standardization

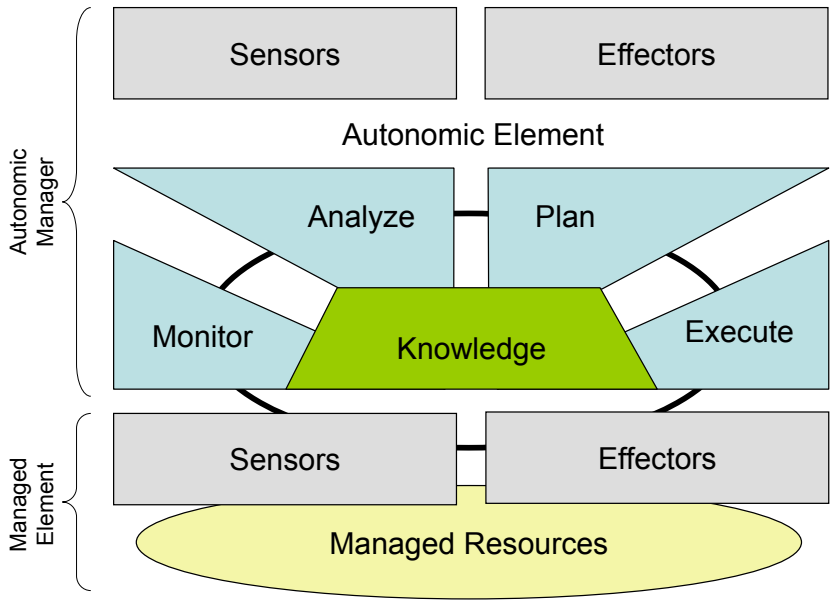

Figure 2.2: The MAPE Control Loop for Autonomic Computing as defined by IBM and introduced in [ibm05]

activities such as the Autonomic Communication Forum [DPF+08].

The ETSI GANA approach [MCR+16] is another standardization activity that aims at providing a unified framework for implementing autonomic behaviors and autonomic network management in general. Thereby, the key elements are given by the concept of a Decision Element (DE) and a Management Entity (ME). A DE is an entity implementing a feedback control loop in a similar fashion as mandated by the IBM MAPE control loop thereby managing an assigned ME. The DEs are structured in a number of layers spanning across the node and network architecture as a whole. These layers are named as follows: 1) *Protocol Level* - control loops embedded within protocol entities (e.g. as in the case of OSPF), 2) *Function Level* - containing Decision Elements managing specific functions within a node, e.g. Routing DE, Forwarding DE, Mobility DE ..., 3) *Node Level* - contains a single Node Level Decision Element overseeing the overall node on top of the Function Level DEs, and 4) *Network Level DEs* - centralized DEs or decentralized overlays of DEs that manage the overall network with respect to required networking functions. Inline with these layers, the ETSI GANA approach [MCR+16] explores ways of integrating Autonomic Network Management with Software Defined Networking and Network Functions Virtualization.

The following paragraph presents examples of applying Autonomic Computing and Networking principles to the emerging 5G mobile network technology as well as to wireless network in general. The GogNet architecture for Network Management featuring cognitive capabilities is designed in [XAY+16]. GogNet is meant to boost the management and control of 5G mobile networks where large amounts of traffic are expected, since periodic sensor data is also expected to play a role in addition to traditional human-to-human communication. This will be supplemented by enabling the connectivity of large number of devices reaching high levels of complexity and traffic volume. Hence, the work of the accompanying CogNet H2020 project aims at mastering the complexity of emerging 5G networks by introducing autonomic mechanisms with respect to organization, config-

uration, security, and optimization issues. This is achieved by introducing autonomic mechanisms in the scope of NFV and SDN, which are described in the coming sections. The mechanisms are based on policies - i.e. a *Policy Manager* component is in place, a *Data Collector*, and a *CogNet Smart Engine* providing possibilities to apply different machine learning algorithms in order to optimize the applied mechanisms and policies. It is worth mentioning that the whole machinery is implemented within the ETSI NFV architectural framework [NFV12]. [NCC+16] is another approach to Autonomic Management for 5G network by exploiting the power of the NFV/SDN trends as described in section 2.2.9.The resulting reference architecture is called SELFNET and is divided into Infrastructure Layer, Virtualized Network Layer, SON Control Layer, SON Autonomic Layer, NFV Orchestration & Management Layer, and Access Layer. The layers are devised taking into account the basic NFV principles as presented in [NFV12] and aim at reducing OPEX and CAPEX for the emerging 5G infrastructure, which is meant to facilitate advanced scenarios such as self-driving cars, 3D video, Machine-2-Machine (M2M) communication and further.The autonomic management of wireless networks is at the heart of [KT16]. The proposition is to implement autonomic network management and policy execution for wireless environments by decomposing the network in a number of layers - i.e. *policy*, *cluster*, *cell* and *user* layers. A hybrid self-learning procedure is presented that uses learning algorithms on the various levels of the network architecture. Thereby, the learning algorithms on different layers influence each other, in order to achieve a global optimum of the undertaken optimization of different KPIs. Indeed, principles from the domain of game theory are applied for that purpose such that an equilibrium of the overall system can be achieved.

Finally, [SC15] reviews approaches to Autonomic Resource Management for cloud computing providers. The paper reviews the historical aspects of Autonomic Computing and advocates for autonomic mechanisms based on the strong usage of virtualization techniques on different layers of the cloud computing stack, i.e. IaaS, PaaS and SaaS. The reviewed techniques for autonomic resource management relate to learning mechanisms combined with resource provisioning and scheduling techniques. The main objective is to introduce autonomic functionalities - i.e. self-configuration, self-optimization, self-healing, and self-protection - in order to support a cloud computing provider in meeting its contractual Service Level Agreements. Thereby, different lifecycle phases of a cloud computing services are analyzed - ranging from Application Design, Workload Scheduling, Ressource Allocation, (QoS parameters) Monitoring, and Self-Management on various cloud computing stack layers. For each of the phases different publications and approaches are listed and referred.

2.2.5 Control Theory

Traditional control theory aims at defining a mathematical model that allows to control a (continuous) dynamic system [HDPT04]. Key constructs are given by the concept of a closed control loop that uses sensors to obtain and communicate measurement data to a controller, which in turn computes some required corrections in order to reasonably regulate the system parameters in question. Thereby, the computation within the controller is based on a so-called *transfer function* [HDPT04]. Various models can be used as transfer functions, e.g. computational paradigms from the area of statistical/machine learning such as Support Vector Machines, Neural Networks or Markov Decision Processes. In many traditional cases, control loops based on simply regulating the difference between measured and desired system parameter values can be quite effective. When regulating this difference, the history of previous measurements and regulative control actions can be taken into account resulting in integral type of transfer functions, such as the pro-

portional–integral–derivative controller (PID controller) [HDPT04]. Furthermore, based on such continuous integral based functions, it is possible to apply Laplace transformations and handle the transfer functions in a mathematic space where various control loop properties can be easily calculated and analyzed. Typical control loop properties of interest are given by *Stability*, *Accuracy*, *Settling Time* - time for reaching the required parameter values after a regulative influence - and maximum *Overshoot* for a targeted value after a control loop regulation [HDPT04]. In addition, various tools are available for modeling feedback control loops in terms of traditional control theory and simulating those, in order to examine the above properties. Well-known tool platforms are given by Modelica [TDW+14], Matlab Simulink [MAT16] and Octave [OCT16]. Research such as [TDW+14] deals with the seamless integration of such tools towards enabling the possibility to efficiently model and simulate the dynamic system in question (e.g. car cruise controller). Traditional control theory is reviewed and relevant concepts adopted for key components of the emerging FDH framework throughout this thesis.

In the following paragraph, a number of recent approaches to control systems' design are reviewed. [Kaw17] introduces a method to controlling dynamic systems based on uniformly hyperbolic control theory. Thereby, the *uniformly hyperbolic control* concept from the mathematical area of non-linear dynamics optimization is combined with traditional control theory, in order to increase the controllability, robustness and practical stabilization features for networked control loops. In that context, a uniformly hyperbolic system - operating in discrete time - is characterized by the split of a complex non-linear system in linear operators with belonging stable and unstable components. This separation allows for better understanding of the dynamic system in question and allows for addressing particular pain points whilst adapting the controller model and improving its overall quality. The book [SPS00] elucidates on optimal control theory that aims to achieve certain goals (i.e. optimize utility functions) whilst at the same time regulating the dynamics of the controlled system. The presentation includes a summary of the main characteristics of traditional and optimal control theory and their generic application to dynamic systems that evolve over time. These systems can be described by different models including continuous time models (i.e. differential equations) as well as discrete time models (i.e. difference/recurrence equations). In addition the different possible constraints, whilst exercising optimal control, are discussed together with the strongly related topic of control stability and correspondingly stability of the overall dynamic system. All these aspects are presented in the light of applying the reviewed concepts to various domains of importance, including *production*, *finance*, *economics*, *marketing* and of course *computer science*. In [FMA+15], the synergies between the domains of software engineering and traditional control theory are investigated. Hence, the authors argue that the software engineering community has proposed various ways to enable software to monitor itself and undertake belonging actions depending on the monitored values. Beyond this, the authors of [FMA+15] argue that concepts from the area of traditional control theory should be extensively used within MAPE types of control loops, in order to extract and design suitable controllers for autonomic entities. Thereby, the reader is guided through the process of controller design and analysis based on steps such as transfer function design, stability analysis, overshoot and settling time goals. Within this scope, different tools from both domains - control theory and software engineering - are combined such as discrete Markov Chains, Linear Regression Analysis, and Event-Condition-Action rules. [CXG17] constitutes a typical research of applying concepts from the mathematical area of non-linear optimization for achieving optimal control of dynamic systems. In [CXG17], a switched non-linear system controller is derived based on a class of functions, called Lyaponov functions. These are special functions with well-defined properties (such as being continuous, having continuous derivatives ...) that - when used for modeling systems of

differential equations for a dynamic system - guarantee the stability of the overall set of differential equations. Hence, [CXG17] models dynamic systems by splitting the overall challenge into smaller ones and switching between different Lyaponov function based models for the smaller parts of the system. [BBM98] proposes a generic framework for deriving systems based on hybrid optimal control principles. Again, in this approach continuous control loops are wrapped in logical decision programs in the sense that a hierarchy of control loops is implemented - a higher level containing a state machine type of model and a lower level controlling the actual entities in place through a continuous loop. Hence, an interacting collection of dynamical systems emerges in such a setup. As a result, a theory for synthesizing such hybrid multi-layer collaborative control loops is introduced. The effort boils down to a generic mathematical model encompassing different sets containing artifacts of relevance, such as *states*, *continuous control models*, *jump relations between different states and continuous models* etc. The proposed framework can capture a variety of mathematical constructs of relevance, such as differential equations, Petri nets, Turing machines etc. A control scheme based on synergetic control mixed with discrete-time linear variation is researched on in [RASM17], in order to cope with potential unexpected disturbances during a control loop operation. An abstract understanding of *synergetic control* explains it as an interdisciplinary field related to non-equilibrium systems, which are influenced in a way as to increase their robustness and stability. Hence, [RASM17] combines recurrence difference equations (i.e. a form of discrete time "differential equations") with synergetic control and applies those to control operational aspects of rotary electrical machines and the temperature of a continuous flow stirred-tank reactor. The application of special online machine learning techniques in the scope of an advanced controller behavior is at the heart of [XHLH13]. The main idea is to utilize online learning approaches from machine learning - such as special types of neural networks or Markov Decision processes - and combine those with sparse kernels. Sparse Kernels are a technique for influencing the preprocessing of the involved data in a way that it is projected in suitable dimensions/spaces thereby minimizing the computational complexity/effort for the preprocessing (i.e. sparse). Indeed, reinforcement learning and Markov Decision processes are used as mathematical models to cope with non-linear control problems using these enhancements.

2.2.6 Policy Based Network Management

In the current section, various available policy management frameworks are reviewed, which are potentially relevant for different components of the emerging self-healing framework. The resulting insights are utilized later on (in the design and specification chapters), in order to specify and implement an applicable policy handling module to be employed for the belonging tasks of the FDH self-healing framework.

CIM : The Common Information Model (CIM) [CIM16] is discussed, developed and standardized at the Distributed Management Task Force (DMTF) [DMT17]. CIM is worked on and enhanced with the ambition to turn into a common standard for the management of ICT based systems. It is intended to provide the means for establishing a unified, vendor and platform independent interface for handling distributed systems and applications. Thereby, CIM addresses the fields of network, system and service/application management. The CIM data model is described in the DTMF's specific Managed Object Format, which is based on the IDL (Interface Description Language) [OMG04] associated with CORBA [HS15]. Beyond these generic considerations, with respect to policies, CIM offers a specially designed policy language called CIM-SPL (CIM Simplified Policy Language) [CIM09], which is intended to provide an easy-to-use basis for the specification and management of whole policy sets. Similar to the overall CIM model, the goal of CIMP-SPL is to

offer a tool for the efficient management of distributed systems, in that case with the focus on policy based management (e.g. automatic reactions, filter rules, resource access regulation ...). The core aspect of CIM-SPL is constituted by the possibility to define *if-condition-then-action* types of policies towards handling various operational challenges in networks and distributed systems (e.g. access control and packet filters). Thereby, during the emergence of CIM, different concepts and understandings were considered, which have already been discussed and established in the scope of IETF. For example, common definitions are taken into account which relate to relevant terminology elucidated by belonging IETF standards [WSS+01]. Thereby, a policy is elaborated as *"A definite goal, course or method of action to guide and determine present and future decisions"*, to mention the concept at the heart of the current discussion.

The information model that embeds CIM-SPL is given by artifacts, which are contained in the existing CIM Policy Model. The latter implies that CIM-SPL related concepts and classes constitute specifications (inheritances) from existing CIM concepts. Indeed, a policy rule in the scope of *CIM-SPL* is derived from the class *CIM_PolicyRule* and is called *CIM_SPL_PolicyRule* [CIM09] The *CIM_SPL_PolicyRule* contains a string attribute, which is denoted as *PolicyString* and stores a policy that is defined in CIM-SPL. Beyond the above considerations, the management of policies within CIM embeds aspects such as *Policy Set*s, *Policy Group*s, *PolicyTimePeriodCondition*s etc. These concepts are essential for organizing and managing the overall set of policies for a system/network.

The resulting framework of CIM based classes together with CIM-SPL facilitates the definition of policies using a (CIM-SPL based) language for boolean expressions and constraints. These are evaluated according to the specifics and properties of a particular context/situation. The evaluation of the boolean constraints results in a decision leading to the execution of management activities on the system/network in question. For more details on the CIM-SPL grammar, the reader is referred to [CIM09].

IETF Policy Core Information Model : The *IETF Policy Core Information Model* is defined in [MESW01]. It aims at capturing the basic concepts for the definition of policies and specifies an extensive policy model, which can be used for implementing a policy based network management solution. Indeed, the model in [MESW01] is an extension of the *CIM-Policy Model*, which was discussed above.

[MESW01] provides a number of definitions and diagrams with corresponding relations among the involved concepts, which constitute the *IETF Policy Core Information Model* as a whole. Thereby, concepts such as *Policy Rule*s, *Policy Condition*s, *Time Period*s, *Policy Action*s as well as means for grouping, organizing and managing policies are introduced. Furthermore, other important aspects such as *Policy Decision Point*s (PDP) and *Policy Enforcement Point*s (PEP) are introduced. Thereby, a PDP stands for a point where a policy is evaluated and a decision is taken, whilst the PEP constitutes the location at which the resulting action is executed, resulting in the enforcement of the policy. Thereby, a PDP and PEP might be residing in the same device, but can also be physically separated, in order to delegate the resource consuming process of policy evaluation to a more powerful node that is not concerned or involved in vital network functions like routing or forwarding of packets. The aspects of PDP and PEP, along with the concept of a Policy Repository, are paramount for the application and deployment of policies for policy based network and systems' management.

ACL : *ACL* [Shi07] stands for Access Control Lists and constitutes a software based approach to managing and regulating the access to IT and network resources. The aspects of resource access are determined by the usage of policies, which can be defined via different policy languages/frame-

works (e.g. Ponder that is presented later on in this chapter). Examples for resources requiring access regulations/restrictions are typically given by entries/files stored in a file system, databases, as well as packet forwarding rules of a router. For instance, within Cisco routers, it is possible to employ ACL concepts [ACL17], in order to block/restrict and regulate the forwarding and routing of (IP-)packets depending on some high level security considerations, which were analyzed in the scope of the management processes for the network/system in question.

COPS: COPS [Her00] is an abbreviation for Common Open Policy Service and is an IETF based framework for implementing Quality of Service, e.g. on top of best-effort[11] networks like Internet Autonomous Systems, thereby efficiently distributing [HRM+01] and enabling the instrumentation of additional QoS policies across the network. In this line of thought, COPS defines the signaling mechanisms and the involved components towards the deployment and steering of a QoS machinery in packet based networks. Key designated COPS components are given by the *PEP* - Policy Enforcement Point - and the *PDP* - Policy Decision Point, which generally acts as a Policy Server storing the policies and evaluating them according to the current conditions and requests. Thereby, COPS utilizes the traditional client-server model whereby the PEP acts as a client and the PDP as a server being requested to evaluate particular policies (e.g. admission control rules and constraints in the scope of RSVP[12]-Integrated Services[13]). For instance, in the RSVP context the PEP is given by the RSPV-Router. Furthermore, efforts - e.g. [BJP05] and [AMB02] - were in place to facilitate the usage of COPS with *DiffServ* [WCW+98], which is the alternative IETF QoS technology to Integrated Services. In the scope of DiffServ, the routers decide on the QoS prioritization of an IP-packet based on information carried in the IP packet header, i.e. there is no pre-reservation of network resources for serving the needs of currently running applications. The integration of DiffServ and COPS typically requires the consumption of monitoring information and the adaptation of the COPS-policies for the involved routers [AMB02].

DEN-ng: DEN-ng is another framework for policy based network management which is applied in the area of network and systems management [Str02] [BDJS10] [Str03a]. DEN-ng stands for *Directory Enabled Networks- new generation* and provides an information model and a policy model addressing various topics relating to the management of networks and systems - e.g. configuration aspects. As presented in [Str02], DEN-ng comes with the following understanding of the term policy - *"a definite goal, course, or method of action to guide and determine present and future decisions. Policies are implemented or executed within a particular context"*. [Str02]. Furthermore, additional aspects of a policy are elucidated as *"A policy condition is a representation of the necessary state and/or prerequisites that define whether a policy rule's actions should be performed. A policy action defines what is to be done to enforce a policy rule when its conditions are met."* [Str02]. Correspondingly, DEN-ng provides the possibility to specify sophisticated Event- Condition-Action policies for the management of IT and network resources. This involves for example the usage and instrumentation of a variety of context information as well as timing constraints such as time periods.

Another important aspect that comes with DEN-ng is the idea of the so-called policy continuum [DJS07] allowing for different views on the management policies for a network. The DEN-ng policy continuum is illustrated in Figure 2.3 with its different views including *Service Level Agree-*

[11] *Best-effort* relates to networks without any or with only minimal intrinsic QoS instrumentation

[12] RSVP [BZB+97] stands for Resource ReSerVation Protocol and is a QoS protocol for guaranteeing traffic and service quality based on the pre-allocation of resources in service provider networks

[13] Integrated Services or IntServ [Wro97] is an IETF QoS framework that defines the principles for resource reservation across a network with the goal to provide guaranteed network characteristics for particular classes of applications and services

ments, *Guidelines* and *Goals* within the *Business View*, and boiling down to the instance specific policy considerations on the lowest level (i.e. the *Instance View*) in Figure 2.3 , which are concerned with aspects such as device specific MIBs (Monitoring Information Bases), CLIs (Command Line Interfaces) and PIBs (Policy Information Bases)[14]. The policy continuum provides a conceptual framework for the understanding of the term policies on different organizational and technical levels of network management. Furthermore, research efforts such as [DJS07] aim at formalizing these views and providing the possibility to manage, author and efficiently analyze policies on different levels.

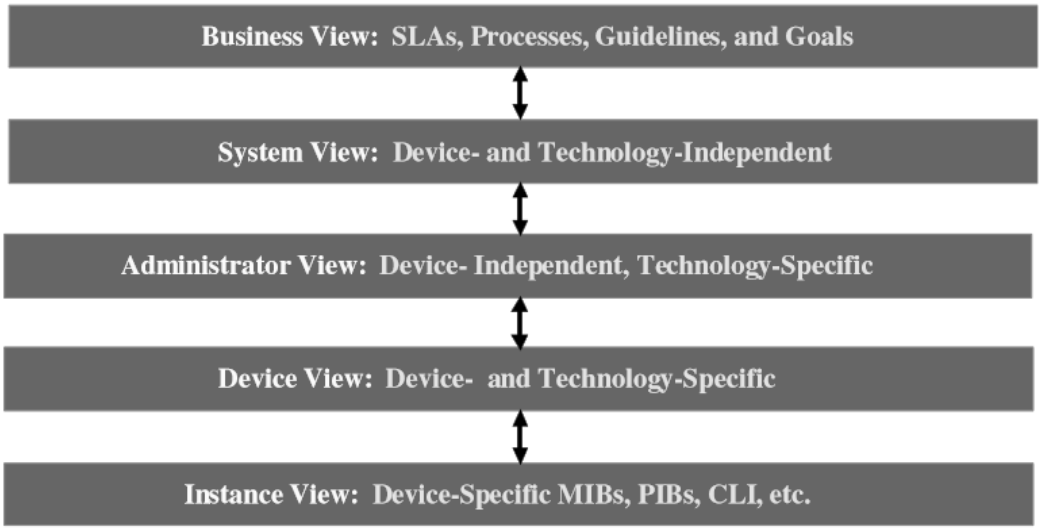

Figure 2.3: The DEN-ng Policy Continuum as depicted in [DJS07]

DEN-ng is strongly related and has largely influenced the SID (Shared Information & Data Model) [SID17b] [SID17a] initiative of the Telecommunications Management Forum [TMF17]. SID and DEN-ng are both based on UML, which implies that the incorporated policy related concepts are UML compliant too.

Finally, it should be noticed that DEN-ng is at the heart of the FOCALE [SAL06] architecture for autonomic networking, which was presented and discussed on in section 2.2.4. More detailed information on DEN-ng can be found in the diverse literature on policy based network management such as [Str03b].

iptables : The next selected framework to present - embedding policy alike concepts - is given by the *iptables* [NET17] tool, which is available within Linux. *iptables* is constituted by a standard Linux program running in user space - i.e. on top of the kernel/core operating system software - and facilitating the definition of filters and chains of rules which can be applied and influence different phases of the packet processing. Key packet processing phases are given by: *PREROUTING* - all packets in a network element/device have to pass this phase that allows for implementing policies before a packet gets dispatched on its way - i.e. before a routing decision for a network interface card, *FORWARD* - this phase is relevant for applying policies to all packets which are forwarded through a device (a router) in the sense that they arrive over a NIC and are pushed to another NIC after a routing decision has been made[15], *INPUT* - the phase facilitates the application

[14] PIBs [SM03] serve as a policy storage entity for the clients in a COPS policy provisioning environment.

[15] This characterization actually implies that packets passing through this phase cannot originate from device local

of policies on packets as soon as they have arrived on a device over a network interface, *OUTPUT* - the phase enables to influence (through *iptables* policy rules) packets which are leaving the device/node, and *POSTROUTING* - all packets in a network element/device pass through this phase after a routing decision has been made. At these phases different actions can be configured for traffic matched by specific filters. Key possible actions are given by *ACCEPT* - a packet can pass, *REJECT* - a packet is to be rejected and an error packet should be sent out, *LOG* - a log entry is created (typically using the *syslog* [SYS17] daemon of the operating system), *DROP* - a packet is ignored and no response is further sent, *REDIRECT* - the destination address of the packet is manipulated as to redirect it according to the requirements of a policy, and *MASQUERADE* - the source address of a packet is changed before the packet leaving the device/node[16]. The variety of the phases and the possible actions allows for introducing a large diversity of packet processing chains and functionality based on packet/flow policy rules. These aspects can play a significant role in the implementation and provisioning of UTM- boxes[17] as well as for functions like NAT (Network Address Translation) [ES01].

XACML : The *eXtensible Access Control Markup Language* (XACML) [OAS17] is another policy description language which is used for access control regulation. It is developed and standardized within the OASIS[18] consortium. XACML is constituted by an XML/XSD scheme that facilitates the definition and processing of authorization policies. XACML aims at establishing common terminology and understanding with respect to access control, as well as common open interfaces, which would positively influence the interoperability of implementations of access control modules. In general, XACML relates to the insights and terminology which is laid down in [VCF+00]. [VCF+00] defines an authorization framework where key relevant aspects are elucidated as follows *"A policy is retrieved by a Policy Retrieval Point (PRP) from a Policy Repository, evaluated at a Policy Decision Point (PDP) or Policy Consumer, and enforced at a Policy Enforcement Point (PEP) or Policy Target."* [VCF+00]. XACML is received as being a suitable framework for implementing the flexible ABAC (Attribute Based Access Control) [HFK+14] method [LZGC08]. ABAC implies that various attributes (small pieces of data) are attached to objects of interest/relevance (e.g. user profile, action specification or resources). These attributes are correspondingly used to determine whether a user has the right to access particular resources or to execute an action. Based on ABAC principles, other access control methods such as RBAC (Role Based Access Control) [RBA10] can be realized by using XACML policy descriptions [LZGC08] [AHCBCD06].

In addition to the above described aspects of access control realization, various deployments and research initiatives demonstrate the utilization of XACML in the scope of different domains. For instance, [She09] discusses on the application of XACML for protecting Web Services. Furthermore, XACML is used for Access Control in Enterprise Networks [CDL+10] based on the TNC (Trusted Network Connect) architecture of the Trusted Computing Group (TCG) [TCG17]. The utilization of XACML towards the *Policy Obligation* based protection of distributed applications/services in the scope of Grid (Computing) and networking security is elucidated in [DKdLS08]. Thereby, the term *Policy Obligation* stands for and is understood as the fact that an *"XACML policy can also specify the Obligations as actions that must be taken on positive or negative authorization decisions"* [DKdLS08], thereby connecting the evaluation result of a policy to the consequent execution of a required action. A final interesting example is given by the application of XACML

processes,

[16] This option can be used to implement NAT functionality within a router node.

[17] *UTM* stands for Unified Threat Management - a concept which implies the implementation of various intrusion detection and prevention mechanisms in one box protecting a network infrastructure from external attacks.

[18] OASIS stands for *Organization for the Advancement of Structured Information Standards* and is a non-profit organization that standardizes and maintains various open formats (e.g. OpenDocument und DocBook).

for the purpose of implementing QoS in (best-effort) IP networks. Thus, the QoS is implemented using the principles of Integrated Services (i.e. IntServ) by utilizing the RSVP [BZB+97] protocol for that purpose. Thereby, the previously presented framework of COPS is used as a baseline architecture within which XACML plays the role of a format for exchanging QoS and admission control policies.

To round up the presentation of XACML related aspects: various frameworks for handling XACML policies have been implemented and are currently available. Examples are given by OpenAz [Ope17a], XEngine [XEn12] and the Enterprise Java XACML [ent12], to mention some of the available XACML frameworks.

Ponder : *Ponder* [DDLS01] constitutes a language for specifying policies, which was developed over a number of years at the Department of Computing of the Imperial College in London. In addition to the policy description language, tools and services were developed that support the Ponder based definition, analysis (e.g. in terms of consistency and conflicts) and deployment of policies. As of the majority of the presented and available frameworks, one of the main goals of Ponder is to provide a powerful and efficient framework for the provisioning of access control rules/policies, e.g. for firewalls, file systems and data storage resources, as well as in the scope of programming languages and execution environments such as the Java virtual machine. Ponder achieves these aspects in a similar way as the other presented frameworks - by providing means for specifying event triggered policies which issue actions based on some contextual/environmental conditions. Thereby, the evaluation of a policy implies the execution of an action, which was already denoted above as a *Policy Obligation*.In the scope of Ponder, the understanding regarding *Policy Obligation*s is similar to the one in XACML and is described as policies that *"...specify the actions that must be performed by managers within the system when certain events occur and provide the ability to respond to changing circumstances."* [DDLS01]. As elucidated in [Pil04], Ponder policies are *"specified at a high-level of abstraction, then broken down internally into simple rules. Finally, the policies are compiled and mapped to rules for the network devices"*. Thereby, the compiled output embeds the specific features/aspects of a device, which are related to policy obligation and action execution, e.g. specific SNMP MIBs or Command Line Interfaces.

Focusing further, beyond the general application of Ponder to access control and to the management of networks and distributed systems, its developers intend for Ponder supporting the monitoring of particular security trends/violations and facilitating the analysis of significant observed/-monitoring information for security purposes. Beyond the basic event-condition-action aspects, other substantial features are for example given by the possibility to group policies depending on their scope, target, or the role of the action executing entity, involved through the policy obligation concept. In that scope, it is also possible to define and group policies in a way such that they provide a kind of library in terms of artifacts, which can be reused in various scenarios, situations and network/system configurations. This enables Ponder to scale up in terms of the size and properties of the managed network /system.

A *Ponder2* toolkit [Pon17] was developed as a successor of the original one thereby using the experiences and redesigning large parts of the initial software components of Ponder[19]. Thereby, Ponder2 claims to extend the scope beyond network and systems' management by providing means for implementing policies that can be used *" at different levels of scale from small, embedded devices to complex services and Virtual Organisations."* [Pon17]. Additional interesting topics around Ponder are for instance given by the fact that different policy models are implementable through the use of the Ponder concepts. For instance, the work presented in [WS04] shows how

[19] In that case, the initial toolkit implementation is meant, which is not any more supported.

the previously presented CIM-Policy Model can be realized by using Ponder.

PBR : Policy based Routing (PBR) [Bra89] denotes the concept of distributing policies across a network, in order to facilitate the control of traffic flows. Such policies can be either employed at the level of inter- as well as on the level of intra-domain routing. Thereby, it is possible to dynamically select the best Internet Autonomous System (AS) to which particular traffic should be forwarded or to steer the traffic load within a particular AS network, in order to optimally utilize the available resources (i.e. routers, switches, gateways, ...). There exist languages for PBR, such as RPSL (Routing Policy Specification Language) [KBA+99], which allow the implementation of rules for the above outlined dynamical steering of traffic. Examples for the implementation of policy based routing are given by the PBR framework [PBR17], which is implemented in Cisco routers, and by the corresponding Unix based tools (e.g. *ipfw* [IPF16]).

2.2.7 Dependable Systems

The concept of dependability and dependable systems has been the topic of many discussions [Ran98] [Lap92] [ALRL04] [LAK92] [Wer07] [Ran98] within the research community. Many of the core concepts (such as fault, error, failure, alarm ..) that the current thesis builds on have been extensively discussed within the *dependable systems community* over the past years. [Ran98] provides a comprehensive illustration of the key aspects related to dependability and to the topic of this thesis. These key aspects are depicted in Figure 2.4 and provide an overall picture of the various items related to the dependability of computing systems. Clearly, the current thesis deals with a limited scope in comparison to the one depicted in Figure 2.4. Thereby, the scope excludes to a large extent the aspects of *Safety*[20] and *Confidentiality*, whilst at the same time dealing with the rest of the topics in Figure 2.4, as discussed in section 2.1.

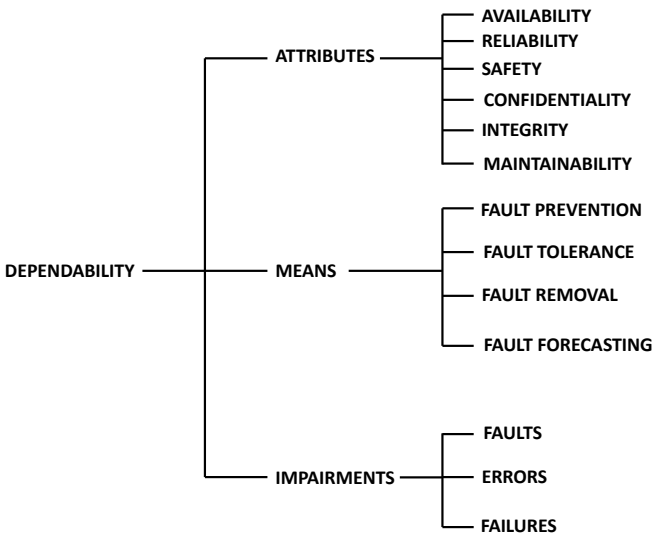

Figure 2.4: The unifying Concept of Dependability as proposed by [Ran98]

Next, a comprehensive overview is provided, related to the topic of dependable systems. In [YFN+14], the modeling of software reliability based on high-order Markov Chains is researched

[20] *Safety* is mainly perceived as the safety of the persons dealing with a computing system.

on. The authors provide a framework for describing different software components with their reliability aspects as well as the probabilities for single module failures and the implication on related system components. As mentioned above, the framework is based on Markov Chains of various orders, i.e. number of failures or component failures influencing directly the failure of another component or of the software in question as a whole. Indeed, the proposed approach allows for theoretically evaluating the performance of a complex software system. The research presented in [HDF16] deals with dependability aspects in the scope of the emerging and widely accepted Software Defined Networks[21]. Thereby, the authors argue that eventough SDN allows for flexible network control and that way contributes to an increased dependability, the dependability of the corresponding SDN controllers itself might become an issue for the overall network or system. Therefore, a framework for assessing the criticality of a variety of involved SDN entities is proposed with belonging models and algorithms for calculating and evaluating their reliability in an SDN context. Indeed, the framework builds strongly on the fact that SDN based networks are mostly virtualized and embeds various entities from the virtual machines domain into the emerging models, such as *virtual NIC*,*Open vSwitch* and *VM Instance* in addition to traditional non-virtualized entities like *physical NIC*, *Linux Bridge* and further. Correspondingly, the relations between the above entities are modeled within appropriate mathematical frameworks, such as graphs and graph entropy setups, thereby executing relevant algorithms (such as PageRank [BP98] algorithms of google) and evaluating the overall SDN dependability, e.g. with respect to the importance/criticality of single entities and the overall system design. In the spirit of dependability modeling, [NBM15] presents a recent development related to the concept of Fault-Tree-Analysis (FTA) towards system dependability description and evaluation. The work encompasses the development of an extended version of FTAs, which is also suitable for dynamic systems. In that context, the term *dynamic systems* stands for systems where the components can have multiple intertwined states (e.g. degradation, overload, ...). Hence, the components are not only in *failed* or *not failed* state but exhibit different states and levels of operations. Indeed, [NBM15] extends the FTA concept by such levels of operations and allows the corresponding dependability assessments. These assessments/evaluations are realized based on suitable mathematical models such as Colored Petri networks[22]. Moreover, [TGSB15] proposes the usage of Continuous Markov Chains for evaluating the dependability of power networks. Based on carefully considered properties of the power line and belonging maintenance procedures, different indicators of interest can be mathematically derived based on the underlying Markov Chain model - e.g. *expected time spent in system failure state* or *expected number of paths/transitions (can be also seen as a possible scenario) leading to an error state*. Such indicators allow evaluating the dependability of the power network in question.

Complementing the above elucidations, the authors of [SWLVH13] present a software based approach for increasing the dependability of production automation systems thereby utilizing so-called *soft-sensors*. Soft-sensors are virtual sensors taking into account different measured values (from real physical sensors), in order to obtain/extrapolate the value of a further KPI. Thereby, the goal is to enable production automation systems to reasonably react to sensor faults along the production line. Typically, production lines are shut down for maintenance in the face of sensor faults and questionable sensor values leading to outages and revenue loss in the production process. Hence, carefully designed soft-sensors and belonging reconfigurations in case of malicious physical sensors bear the potential to decrease downtime and increase revenue at the same time.

[21] Further details on Software Defined Networks and related concepts are given in section 2.2.9

[22] Traditional FTA approaches are based on Markov Chains for evaluating the overall system dependability with respect to the captured fault-tree and belonging relations among the components.

Therefore, [SWLVH13] proposes a model based approach to the design of soft-sensors considering different aspects such as the analytical dependencies between physical sensors, process/machinery constraints, as well as the precision of the involved physical sensor devices. The dependability of Industrial Control Systems in terms of security is discussed in [KvdHK+14]. [KvdHK+14] deals with the definition of the research direction and challenges' identification as a whole. The main insight is that industrial control systems focused mainly on safety hitherto, whilst the topic of security was largely neglected. Just recently, the relevant industry and research community realized that industrial automation is increasingly being realized by highly distributed architectures often connected using traditional Internet type of communication technology. Hence, typical dependability and security issues emerge relating to aspects such as cyber security, integrity, confidentiality, availability, the need for reliable and secure update processes as well as typical management processes/tasks such as Fault-Management (e.g. inline with incident response). In a similar scope, [EBG+15] deals with the dependability of wireless sensor nodes, which are used for proactive maintenance and monitoring in industrial systems. In this line of thought, different issues related to WSN are handled and reviewed followed by a discussion on typical techniques for WSN based health management in industrial systems. Thereby, the health management is highly dependent on the dependability of the WSN and the possible issues which can emerge and need to be handled, such as timing problems (w.r.t. to sensors and measurements), security issues (e.g. DDoS attacks, Man-in-the-Middle, Physical Attacks ...), operational challenges such as battery lifetime optimization and further. In order to minimize the effect of such challenges and properly analyze them, different techniques are reviewed and brought to the reader's attention, such as statistical approaches, Hidden Markov Models, model-based approaches and further. The general dependability of embedded systems is at the heart of [GFN15]. Thereby, the aspects of security and data privacy/protection are addressed along with classical aspects of dependability, such as MTTR (Mean Time To Repair) and MTTF (Mean Time To Failure). Indeed, a methodology is developed that incorporates the utilization of various metrics (i.e. multi-metrics approach) for the overall system evaluation during the design phase and run-time as well. Privacy, security and dependability metrics of relevance are defined and taken into account, e.g. on code, function and system levels, or users' location privacy. The multi-metrics approach allows for prioritizing different aspects, in order to specify the desired form of dependability for the system in question. [EdAG15] argues that the improved hardware design for embedded systems (achieved in the recent years) brings a variety of new features such as logic density, reduced power consumption and increased processing power. However, this comes at the cost of increased sensitivity of the logic circuits and leads to so-called fugacious[23] faults, which can be seen as an indication of imminent failures of an embedded system (e.g. due to hardware issues). In response to that, [EdAG15] proposes a VLSI design scheme for early detection of such faults towards increasing the dependability of modern embedded systems. Thereby, various VLSI code artifacts are proposed in order to detect corresponding faults as close as possible to the level of logical circuits. The design of the dOSEK dependability-oriented kernel for a real time embedded operating system is the main topic of [HLDL15] and [SBD+17]. dOSEK is aligned to the OSEK/AUTOSAR [AUT17b] standard from the automotive domain. dOSEK is in its core a software based concept/component for reliable computing on top of unreliable hardware. The unreliability of the hardware may arise due to various specifics of modern designs, such as shrinking sizes and operating voltages (more such aspects were mentioned above), leading to transient failures, e.g. due to bit flips caused by the chip structure and belonging magnetic fields. Hence, an operating system kernel such as dOSEK would try to detect and avoid such type of hardware failures. Thereby, a special type of *fault*

[23] The fugacious faults in that sense stand for short disturbances in electric/magnetic signals.

avoidance is realized, which is encoded in the dOSEK kernel (on top of a particular hardware platform) during the software development process. Therefore, some key design/coding principles are established such as *"Minimize the time spent in system calls and the amount of volatile state, especially of global state that is alive **across** system calls"* [HLDL15], *"Avoid indirections in the code and data flow."* [HLDL15], and *"Assess the actual impact of any dependability measure early and often."* [HLDL15]. These basic principles are complemented by a continuous integration with fault-injection testing for various HW specific problems as well with a completely static design (e.g. well-known size of objects at compile time).

With respect to quality assurance, tools and processes for dependable systems design; [MMS+17] presents the Mercury modeling environment that is especially focused on the topics of dependability and performance of general systems. It allows for creating models - related to dependability features - of systems and subsequently evaluating their dependability based on the derived models. The mathematical foundations of the deployed models are given by a variety of mathematical/algorithmic frameworks such as Stochastic Petri nets, Continuous Time Markov Chains, and Energy Flow Models, to give some examples. The research presented in [GKT13] constitutes another exciting work related to the topic of systematic fault-injection testing, in order to evaluate systems' dependability. Thereby, the EDFI tool[24] for designing effective general purpose fault-injection experiments is presented. The authors argue that existing techniques and static presumptions applied in the course of traditional code instrumentation - such as presuming static memory locations or random fault-injection - constitute shortcomings that must be overcome. One possible approach is to facilitate fault-injection during the run-time execution and inject the faults in an event driven manner, which goes beyond the usual relying on code instrumentation but influences the execution behavior with its underlying specifics as a whole. Moreover, [TÇE17] investigates the usage of Colored Time Petri nets for modeling FTAs and evaluating the dependability of systems consisting of multiple autonomous agents. Based on the required collaboration of the involved agents towards the completion of a complex goal, the different tasks of the agents are modeled by colored tokens in Petri networks with a time measurement extension, such that the overall system can be simulated and its dependability improved. A survey on dependability assessment and testing techniques based on fault-injection is provided in [NCM16]. The goal of [NCM16] is to support research, development and industry in selecting an appropriate technique whilst striving to test and ensure various dependability aspects of networks and systems in general. Some of the techniques which are elucidated and compared in [NCM16] are given by: mutation testing, state models, Markov Chains, FTAs, dependability benchmarking, architectural approaches to fault-injection as well as various types of faults, which are to be introduced for dependability testing, such as pointer errors, initialization errors, bit flips, code changes, interface or error injection. The research in [SMTZ13] deals with the dependability of cloud computing system deployments in case of natural disasters. Therefore, a highly dependable cloud infrastructure is to be deployed in multiple geographically distributed data centers. This implies that aspects such as state/information or VM migration need to be modeled and evaluated in order to assess the overall dependability subject to times/delays and distances among the involved data centers. The proposed modeling and evaluation approach [SMTZ13] incorporates modularly composed dependability descriptions/models - based on so-called RBDs (Realiability Block Diagram) - which are evaluated using Stochastic Petri nets.

In [cL08], the European ReSIST Network of Excellence presented its view on *Resilience* and its relation to the concept of *dependability*. The elucidations are in line with the views presented in this thesis. Thereby, key properties and relations among *dependability* and *Resilience* are estab-

[24] EDFI stands for *Execution-Driven Fault Injection*.

lished by requirements such as the *"persistence of service delivery that can be justifiably trusted, when facing challenges"* [cL08], the proactive avoidance of imminent failures, as well as the *"persistence of dependability, when facing challenges"* [cL08]. In order to fulfill these requirements, different design principles are proposed for achieving Resilience, e.g. *adaptivity*, *evolvability* (i.e. the capability to deal with various types of long-term changes), and *assessability* in terms of offline/pre-deployment verification and evaluation. [HBZ+14] proposes the concept of Multi-Layer Dependability and correspondingly advocates for designing dependable systems based on a cross-layer approach throughout the utilized hardware and software stacks. Thereby, the idea is to adapt the different hardware and software layers to each other such that error propagation across various layers and components can be avoided and erroneous states can be contained before leading to a large scale damage. In this line of thought, various layers are conceptualized and taken into account such as *Device and Circuit Layer*, *Compiler and Synthesis Layer* as well as *Runtime System Layer*, to name a few important layers.

The realization of dependability features through semi-automatic feedback control loops is the topic of [FHH14]. Thereby, the semi-automatic aspect is especially addressed in [FHH14] by providing a "human in the loop" framework with comprehensive user interfaces that allows to involve the operators of large scale software landscapes in the required maintenance routines and reactions of the software components towards increased overall dependability. Furthermore, [PvHG14] comes up with an approach to dependability (for component based software systems) by utilizing online Failure-Prediction. For that purpose, different types of models are considered and designed, such as an architectural model of the involved hardware and software components capturing the dependability among them, a prediction model for each component containing the component's state and probability for a failure, as well as a prediction model for the overall system in question. Hence, Continuous Markov Chains are utilized as an underlying mathematical model for online failure management and prediction. The publication [CdLL+17] deals with robustness and dependability for self-adaptive software systems, i.e. software systems implementing autonomic control loops such as IBM MAPE. The goal of the proposed methodology is to detect design faults whilst evaluating the robustness of the control loop in question. For that purpose, mutated input probes are used as input for the control loop in a simulated environment. This leads to a robustness testing framework for autonomic control loops (of software systems), which is applied on a case study related to server load-balancing in data centers.The authors of [GVCR17] initiate a discussion on open problems and corresponding solutions for software based dependability in distributed systems[25]. In that scope, the focus is set on real-time aspects, security and software forensics, as well as adaptive cyber-physical systems. Some of the identified problems and solutions are given by the *need for high quality system models*, *techniques for verification and validation*, *techniques for secure real-time operation*, as well as *the management of inherent dynamic behaviours*, to name a few.

2.2.8 Agent Systems

A great deal of research was conducted in the area of agent based systems during the past decades. Thereby, the initial idea was that software agents will dominate every aspect of our digital society and will be as present as the WWW in people's everyday life. Therefore, different application domains and various types of agents were investigated, which has led to a large amount of concepts and definitions as well as scenario identifications for agent based systems. For instance, [JW96] discusses on different application areas for software agents, including Personal Informa-

[25] The discussion is based on contributions to a special issue of the Elsvier Journal of Systems Architecture.

tion Management, Electronic Commerce, and Business Process Management. In order to fit to the envisioned domains, different key properties are required for the concept of a software agent. Indeed, an agent is viewed in many ways, e.g. as a permanent software component that establishes human-like relations with other agents/humans thereby simply conducting tasks, which a human might fulfill for another human. In that context, software agents are dedicated to a specific task and are able to control their own decision making and acting by monitoring their environment/situation and trying to achieve some pre-defined objectives. Furthermore, the software agents are classified as *gopher* agents, *service performing* agents, and *predictive* agents [JW96]. The gopher agents execute simple straightforward tasks based on pre-defined policies. Moreover, the service performing agents execute more complex tasks in a narrowed domain, which constitute a service for humans or other agents. Finally, the proactive agents perform various services without being explicitly asked and proactively deliver results to other agents or to involved humans. Thereby, the software agents are required to posses two characteristics based on their task or type, i.e. Responsiveness and Proactiveness, which are both at the heart of an agent's behavior. A wide overview of the taxonomy of autonomous agents is also provided in [FG97]. Thereby, eleven definitions of the agent concept are cited coming from different research and commercial initiatives, including companies such as IBM. The commonality between all these definition is that an agent is an entity that can observe its surrounding environment and can proactively act or undertake adjustments upon environmental changes based on pre-supplied goals. According to [FG97], to describe an agent five aspects of its operation need to be captured: 1) environment, 2) sensing capabilities, 3) actions, 4) drives or preferences also encoded in the agent's goals, and 5) action selection procedures. Especially for autonomous agents, it is important to emphasize that they execute concept behaviors based on their own agenda without any human intervention during the operational phase. Furthermore, an interesting classification of agents is provided including *reactive*, *autonomous*, *goal-oriented*, *learning*, and *mobile* agents to name some of the most prominent classes. In the course of this, it is important to mention that one agent can comply to multiple classes at the same time, i.e. it can be reactive, autonomous and learning whilst striving to achieve its goals. In addition, the author of [Jen00] discusses on various advantages of agent-based software engineering. Hence, different definitions of the software agent concept are reviewed and a suitable one is selected and embedded into a canonical view of an agent based system. The selected definition mainly establishes an agent as a software entity that can be placed in an environment and can react in a flexible autonomous way towards achieving its pre-defined objectives. The belonging canonical view of an agent based system frames the environment, in which the agents operate, and their peering and hierarchical relationships, including organizational grouping and interactions. Furthermore, the spheres of visibility and influence (i.e. domains of operation) are defined as a concept, which is of great importance for the design of agent based systems. Moreover, different advantages and disadvantages of agent based software engineering are identified [Jen00]. Some main advantages are given by the agent oriented decomposition of complex systems and the separation of complex tasks into multiple goals linked to the above areas of visibility and influence, as well as by the possibility to implement flexible management of changing organizational structures among the involved agents. The main downside is provided by the inherent unpredictability of agent based software systems, given the possibility for emerging chaotic behaviors of such systems. Hence, one of the main challenges for agent based systems is to ensure the overall stability despite disturbances and potential chaotic reactions of the involved agents. This challenge is also addressed for the emerging self-healing framework in the course if the current thesis.

The communication between software agents is one the key aspects when designing an agent

based system. Different communication paradigms and protocols were used within various communication and technological stacks - e.g. special control protocols such as ICMP or different types of TLV[26] encoded messages (as within SNMP). In that context, the pioneer research work [GK94] discusses some general considerations and proposes belonging principles regarding an Agent Communication Language. Thereby, agents are again considered as larger software entities running in different threads (however in the same memory space) or in different operating system processes (i.e. in different memory spaces). The proposed concepts include components and intermediate agents such as *Wrapper*, *Transducer* and *Rewrite* modules that enable the exchange between federated clusters of agents. The exemplified communication is captured in textual form following some formal grammar rules, which enable the parsing and interpretation of the exchanged messages. [QMSW17] provides a survey relating to achieving consensus among the involved agents in a distributed multi-agent system. Indeed, different types of consensus set-ups might be occurring, like consensus based on *environmental constraints* (e.g. actuator saturation or erroneous states), *event based consensus* where the agents need to converge to a common strategy after being triggered by events, set-ups in which the involved agents can be either *cooperative or competitive by design*, *consensus among heterogeneous agents* operating based on different controller models, as well as *consensus according to different economic goals and constraints*. For each of these consensus set-ups, appropriate (mathematical) models are reviewed and selected that provide a framework for converging to a common behavioral strategy given that the agents have been accordingly designed, as to follow the communication rules of the corresponding model. The models have strong relation to the area of traditional control theory as described in [HDPT04]. In a similar scope, [DL13] investigates the topic of multi-agent hybrid control for smart microgrids. Thereby, the goal is to maintain the required voltages for an optimal operation with respect to economical and environmental aspects. The involved agents are organized in three layers: 1) lower level unit control agents - controlling the operation of involved energy sources and grid distribution elements, 2) middle level coordinated control agents - establishing cooperative behaviors among the lower level unit control agents, and 3) an upper level energy management agent - taking care of higher level energy optimization strategies. Indeed, the different layers utilize various types of models, in order to achieve their goal thereby implementing a hybrid control approach with respect to the overall system. Traditional control theory is applied to the lower level unit controlling agents, whilst the upper layers employ Petri nets (middle layer) and multi-objective optimization for selecting the optimal energy optimization strategy within the upper level management agent. The event-triggered hybrid control of an energy smart grid - utilizing the Internet for control signaling messages - is at the heart of [DYHG17]. The main goal in that context is to reliably meet the load demand with appropriate quality thereby dispatching energy according to the dynamics of the energy network. Indeed, the increasing utilization of renewable energy sources can only be successful, if appropriate load balancing and belonging required communication is implemented. A possible realization on top of the global Internet is provided in [DYHG17] based on a multi-layer architecture encompassing various agents operating based on hybrid control principles. Therefore, Petri nets are again utilized for the automata and communication/collaboration part of the system, whilst the agents underneath implement multiple cascading optimization control loops that are triggered upon an event communication related to shifts in the energy load balance.

Regarding advanced agent based technologies and applications; [Pas16] introduces the Provalets mobile agents. which constitute a new paradigm for rule based software agents that access data based on pre-defined constraints, and further conduct semantic processing and inference based on the Prova rule language [PRO13] and belonging processing engine. Thereby, the Provalets are

[26] TLV stands for Type-Length-Value.

dynamically deployed as micro services into container environments such as OSGi and Docker. [BCE+16] uses agents for implementing autonomic network management by utilizing MAPE control loops on top of a semantic infrastructure, i.e. a set of ontologies allowing for reasoning and decision making. Furthermore, the ontological approach is combined with machine learning techniques - to improve the ontological models - and with SDN/NFV[27] control structures, in order to efficiently realize the autonomic control functions. The authors of [GL16] implement a multi-agent based system for supply chain management, which utilizes and processes large amounts of information (i.e. data analytics is correspondingly applied). Thereby, the system is meant to act similarly as traditional decision support systems that fuse large amounts of information and extract and propose corrective actions. Therefore, the focused supply chains need to be agile and adhere to certain principles such as flexibility and responsiveness. In that scope, different agents are responsible for managing aspects such as orders, procurement, production planing and further. The extracted actions can be either suggested to the humans managing the supply management, or can be automatically triggered when possible and appropriate, e.g. additional orders can be automatically issued. Finally, an interesting application of multi-agent systems is given by the utilization of agents for large scale distributed simulations of real world problems as researched on in [RHLP16]. The authors of [RHLP16] argue that the dynamic systems' simulation based on differential equations (and/or difference equations) does not always explain the system in the desired detail, given that the underlying differential equation models are not capable of capturing every aspect in detail. Hence, a more detailed approach is possible if software agents are made responsible for the simulation of a minimal part of the overall system thereby behaving according to some basic rule of relevance for that system part. In addition, the results of the agents are combined as to extrapolate and simulate the behavior of the overall system in the belonging situations. [RHLP16] provides a comprehensive survey of the variety of agent based simulation platforms, e.g. D-MASON [DMA16], Jade [JAD17], and Flames [FLA13].

2.2.9 Software Defined Networking and Network Functions Virtualization

The following sections describe the development and research efforts from the domains of Software Defined Networking and Network Functions Virtualization. The flow is initiated by presenting the concepts and technologies around the topic of Software Defined Networking, including aspects as Open Flow, network apps, programming languages, and quality assurance for Software Defined Networking. Subsequently, the key topic of Network Functions Virtualization is presented, and its combination and synergies with the idea of Software Defined Networking is discussed in more detail.

2.2.9.1 Software Defined Networking

As a consequence of the above described efforts to improve network and systems' management and control, the concept of Software Defined Networking (SDN) has drawn the attention of industry and research during the past years. The cornerstone of SDN is given by the notion to have an abstract architecture where the data traffic (i.e. data plane) - passing through (or reaching) a network node - is separated from the control traffic and functions (i.e. from the control plane) [ONF12]. The ONF (Open Networking Foundation) [ONF17] white paper [ONF12] discusses on key characteristics of Software Defined Networking. The ONF abstract view on SDN is depicted in Figure 2.5 that was adopted from the corresponding white paper [ONF12]. On the bottom

[27] The SDN/NFV concepts are discussed in the coming section.

part of Figure 2.5, a set of network devices/nodes are depicted that belong to the infrastructure layer, responsible for forwarding and routing of data traffic. These network devices/nodes provide standardized interfaces, which can be utilized in order to re-program and/or re-configure the nodes/devices (and correspondingly the network segment) based on logic implemented within the entities in the above control layer. The control layer can accommodate a large variety of SDN based control modules for realizing different network functions. This may include functions such as routing, QoS management, access control, security, packet inspection, etc. The disruptive approach - as compared to traditional networking - is constituted by the fact that within SDN the control logic is not any more placed inside the network elements, but is instead delegated to a logically centralized component, which oversees the overall network segment and exercises control over all the nodes in question. This is indeed a substantial change as compared to legacy network elements/nodes/devices, where functions such as routing (e.g. OSPF, RIP, ISIS, BGP ...) or QoS (e.g. IntServ or DiffServ) are realized within the devices in a vendor specific manner.

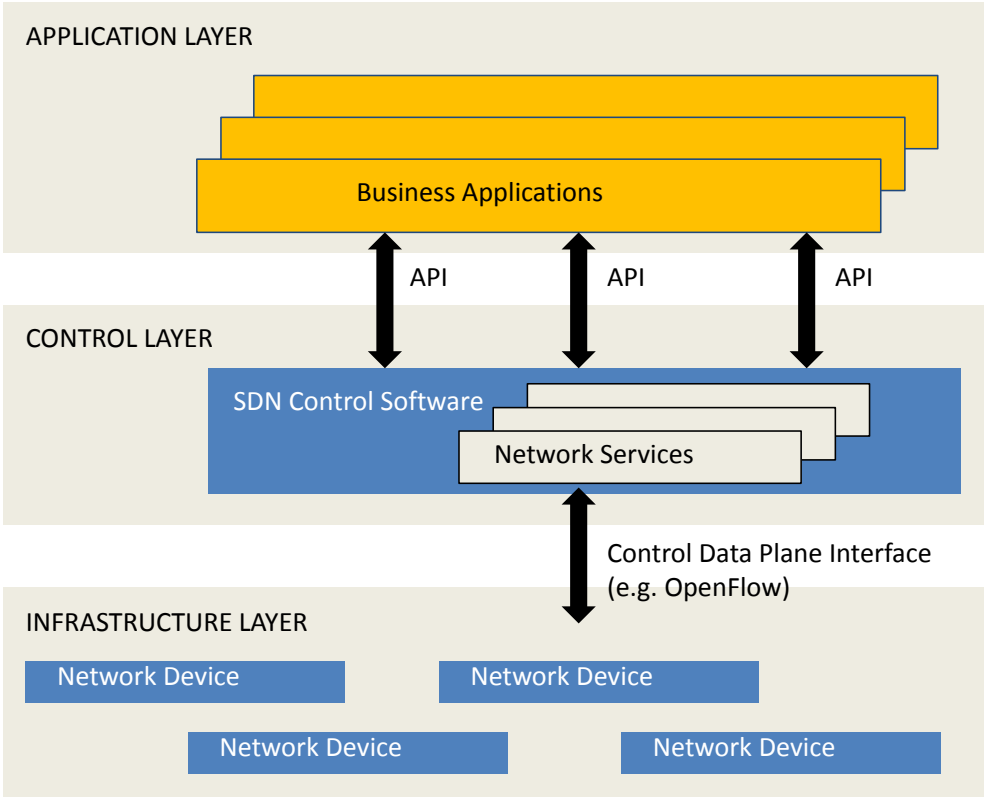

Figure 2.5: Abstracted Software Defined Networking Architecture as presented in [ONF12]

As mentioned above for the legacy approach to network management/control, the networking functions (e.g. routing, QoS, traffic shaping, security ...) are normally realized in a vendor intrinsic interpretation of relevant standards. This becomes a circumstance when one tries to manage these functions in association with different vendor's devices. Thereby, the management interfaces (e.g. CLIs, SNMP MIBs/OIDs, NETCONF, REST, Remote Procedure Calls, ...) are so diverse that difficulties and interoperability issues easily emerge. With the goal to improve the situation around above cumbersome circumstances, SDN requires the network devices to be simple forwarding nodes, which provide standard unified interfaces/APIs. By using this APIs different

networking functions can be programmed/implemented and realized by SDN controllers - i.e. by so-called networking apps. Hence, the control functions - routing, QoS etc. - are delegated and outsourced to a logically centralized layer on top of the infrastructure layer in Figure 2.5. This setup facilitates the fast implementation and introduction of new network services and business applications - refer to the application layer on top of the control layer in Figure 2.5. Indeed, in the scope of SDN it should ideally only be required to implement the logic of the particular network service and application on top of the abstracted API of the involved network nodes. The layered illustration in Figure 2.5 outlines an ecosystem of SDN control software packages and business applications thereby establishing parallels to the concept of apps, which has enabled massive innovations in the area of web and mobile computing.

In recent years, OpenFlow [Ope17b] has emerged as a first standard towards the implementation of the above described SDN concepts. OpenFlow is being worked on in the scope of activities at the Open Networking Foundation [ONF17] and has been first proposed by researchers that were looking for possibilities to test experimental network architectures and behaviors in real world network environments. Thereby, the initial proposition for a clean separation between data and control planes was made as an enabler for handling the experimental traffic (for research trials) separately from the production traffic of real university campus networks [MAB+08].This can be easily realized based on SDN concepts where a controller would look for the experimental traffic passing a router, analyze it and let it pass, whilst the real user packets remain untouched and the network can continue serving its purposes. The basis for the proposed OpenFlow architecture and approach was given by concepts, which were initially laid down in the scope of the Ethane [CFP+07] network architecture for the efficient management of Ethernet switches in enterprise networks. Ethane can be seen as a first step towards SDN and increased programmability of network devices and networks as a whole, thereby establishing a centralized layer that steers the overall set of Ethernet switches in question. Coming back to OpenFlow, it facilitates the dynamic control of different facets of packet forwarding, e.g. path setup through the network, packets dropping according to particular policies, etc. In addition, a reference implementation [Ope11] of OpenFlow is in place, which is provided by the Stanford University in the USA. With respect to combining virtualization and SDN, the Open vSwitch (Open Virtual Switch) project [Ope16] has developed a platform for establishing virtualized programmable networks thereby using different hypervisor technologies such as KVM, VirtualBox etc. These OpenFlow implementations provide a solid set of tools for experiments and trials with open source Linux based routers.

As OpenFlow was evolving and maturing as a standard during the years, a number of key vendors (e.g. Cisco, HP, Huawei ...) started providing OpenFlow interfaces within their devices. This automatically reveals and leads to the potential for developing different types of SDN control software (see Figure 2.5). Such SDN controller software normally comes with a framework/libraries for the fast implementation of OpenFlow based network services. Thereby, the SDN control software facilitates the interaction with the OpenFlow interfaces of the network devices by providing client side APIs. These controller frameworks enable the realization of services in different standard programming languages. The NOX SDN [GKP+08] [NOX17] framework provides means for implementing C/C++ based controllers thereby utilizing the OpenFlow interfaces of a network device. POX [POX17] is a Python derivative of NOX and is based on the core components of the NOX implementation, in order to facilitate the development of platform agnostic SDN controllers. The Trema [SJ17] [Tre17] OpenFlow framework allows for implementing OpenFlow based network services/applications in C and Ruby. The Beacon SDN Controller [Eri13] [Bea17] software constitutes a platform for the Java based implementation of OpenFlow controllers. Floodlight [MA17] [Flo17] stands for another advanced SDN Java based controller, which can be utilized

to exploit OpenFlow interfaces in conjunction with Open vSwitch, as well as to steer a variety of non-virtualized (i.e. physical) industrial devices. A further Java based SDN controller software is given by Maestro [Cai12] [Mae17] - an SDN platform which is optimized to exploit parallelism thereby improving the traffic handling on the level of single traffic flows. It is worth mentioning that Maestro is especially optimized to run on multi-core systems, in order to efficiently control the flows within an OpenFlow network. Finally, an example of a remarkable network service based on the NOX SDN controller framework is constituted by RouteFlow [Rou17] [NRS+11]. RouteFlow realizes various traditional routing protocols on top of an OpenFlow network. Therefore, the topology of the network in question is emulated in a MiniNet [MA17] [Min17] [LO15] emulation environment. Furthermore, the traditional routing protocols (e.g. BGP, OSPF, RIP ..) are setup to run in the emulated copy of the real network. In addition, the events and decisions of the routing protocols from the emulation are intercepted and used to setup the forwarding tables of the network devices in the real network, controlled by an OpenFlow service/app interacting with the MiniNet based emulation and the real devices at the same time. The application of OpenFlow to wireless mesh networks (among others) constitutes the main research activity of [DKB11] and is also reviewed in [JK14]. Thereby, NOX is used to realize an OpenFlow based mobility solution for the migration of IP addresses in case a client changes its position with respect to the mesh access/entry point. Furthermore, [KAMH17] presents various approaches to the application of SDN concepts and technologies to wireless sensor networks. Thereby a number of aspects such as architectural considerations, protocols/standards, applications and security are discussed on and challenges such as interoperability, routing, energy, mobility and localization are put into focus. In the course of the discussion, a special emphasis is set on the specific role of SDN for improving the performance and management of wireless sensor networks. Next, various research efforts are presented which are related to the design, specification, quality assurance and testing of SDN based networks and services.

With respect to approaches to SDN controller design, specification and development: [FGR+13],[MHFv16] and [FKM+16] elucidate on and propose different features for a programming language to facilitate the efficient programming of SDN controllers utilizing OpenFlow interfaces. The major factors to be considered are given by aspects such as traffic monitoring, forwarding policies/behaviors/rules and corresponding updates in case of changes in the network environment. These considerations result in the design of special languages for SDN, which constitutes the main goal of the Frenetic project [Fre17]. Furthermore, the utilization of logic programming for the development of OpenFlow controllers is researched on in [LWN+15][KRW12]. These discussions lead to the proposition of a programming language called Flog, which facilitates the description of decision making strategies based on dynamically received events from the data plane of an OpenFlow network. The challenge of implementing modular SDN controllers is the main topic of [MRF+13]. Based on the Python programing language, the Pyretic domain specific language for building modularized SDN controllers is proposed. Hence, the dynamic composition of controllers based on various abstractions is possible. Such abstractions are given for instance by an abstract packet model or topology, which a specific SDN module has the capability to operate on. [SLBK13] deals with the application of OpenFlow and SDN for the purposes of Cyber-Physical Systems. Indeed, the main contribution of [SLBK13] is constituted by the investigation of approaches to the verification of various critical aspects of SDN and OpenFlow controllers. This verification is achieved by the design and utilization of a specification language called VML (Verificare Modeling Language), which is based on a combination of different formal modeling techniques towards the design of verifiable OpenFlow controllers.

With respect to approaches to quality assurance for SDN controllers: [CVP+12] presents and pro-

poses a framework - called NICE [NIC15] - for OpenFlow controllers testing. Thereby, NICE utilizes methods from the domain of model checking, e.g. state space exploration in combination with the execution of the controller code augmented by additional data (e.g. debug data inserted during compile time), in order to test and find potential errors in an SDN controller. The methods are applied in the scope of NOX based controllers and turn out to effectively increase the controller quality within the experimental setup. The non-functional requirements of consistency and scalability - mainly for SDN controllers updating the forwarding rules/policies across a network - are the topic of [RFR+12], [RFRW11] and [HMR+15]. Thereby, the forwarding rules' consistency is guaranteed on packet and flow level, i.e. models are designed which ensure that whilst the forwarding rules/policies are updated across the network, each packet is handled by only one enclosed/self-contained set of rules within the whole network. Finally, Hedera [AFRR+10] is another impressive research effort related to the design and implementation of an OpenFlow controller for data centers. Hedera explicitly deals with the specific operational constraints that are present within a data center. This includes topics such as energy consumption and special traffic patterns resulting from common data center software/services.

2.2.9.2 SDN in combination with Network Functions Virtualization

Given the basis of SDN as described above, it is a natural extension to apply SDN in conjunction with the virtualization of network nodes and links, i.e. with virtualized networks as a whole. Thereby, the community has introduced the term Network Functions Virtualization (NVF) for the concept of providing network functions such as routing, QoS, mobility, NAT, IDS, IPS etc. in the form of virtual machines/appliances. Once being virtualized, these functions are often referred to as Virtual Network Functions (VNF) whereby the implementation is based on using a combination of SDN with the power of hypervisor virtualization platforms. [HGJL15] discusses on key aspects of NFV and reveals the potentials for innovation, as well as the challenges which emerge with the extensive application of virtualization and NFV. Thereby, [HGJL15] summarizes the fact that NFV decouples the software aspects of a networking function from the underlying hardware and enables the dynamic composition of customer tackled network segments and services. Potential use cases are identified including the virtualization of home and mobile core networks, e.g. the Evolved Packet Core (EPC). The key technical requirements and challenges to the NFV approach are listed thereby encompassing aspects such as performance, manageability, security, reliability and stability. Further research and innovation challenges - related to the identified technical requirements for NFV- are constituted by need for efficient methods for calculating an optimal embedding/placement of VNFs and virtual networks on top of the physical hardware, the migration of virtual appliances in case of hardware malfunctioning, and the outsourcing of VNFs to third parties. The reliability challenge for NFV is researched on in [CSI+14]. Given the fact that the "softwarization" of networking functions naturally leads to reliability discussions similar to those in traditional software and distributed systems approaches, [CSI+14] strives to develop a reliability assessment framework for NFV. The main focus is set on understanding the risks related to NFV reliability and the possible mitigation strategies for containing the impact of faults specific to the virtualization technology in question. In that line of thought, the NFV ISG [NFV12] of ETSI with its *NFV Infrastructure (NFVI)* ecosystem is presented and key architectural artifacts are introduced, such as *Virtual Network Container Interfaces* - enabling the network virtualization on top of the underlying hardware, or *Compute Container Interfaces* - the interfaces for virtualizing the computing power of the underlying hardware. Hence, various reliability requirements and challenges - e.g. hardware faults - for these architectural artifacts are identified and fault-injection

techniques for reliability testing and assessment are proposed. The emerging set of techniques is evaluated in an industrial project involving a telecom infrastructure. Moreover, the authors of [WRH+15] investigate and provide a solution for the NFV issue of performance. This is conducted in the scope of a system called SDNFV, which is proposed by the authors, and constitutes an effort for seamless integration of SDN and NFV principles[28]. The main tools - proposed by [WRH+15] - for improving the performance of virtualized networks are given by *zero-copy* procedures for packet transfer between real hardware and the virtual machines, utilizing non-uniform memory access (NUMA) for efficient process scheduling, exploiting multi-core CPU features as well as the advantages of modern network interfaces with optimizations for hypervisors (e.g. with respect to *zero-copy* procedures). Another approach for improving NFV performance is given by [GLD+14]. Thereby, the focus is on middleboxes providing functions such as NAT, DPI, IPS/IDS etc. The performance improvement is achieved through a consolidated framework that integrates the hypervisor (OpenStack, KVM ..) with an FPGA (i.e. hardware) based approach to implementing the required VNF for the middlebox in question. The overall approach is based on the above mentioned NFVI architecture of ETSI.

In the previous paragraphs and sections, a number of standardization activities were mentioned. [CMS+14] is an approach to bridging the various activities in the areas of autonomic networking, SDN and NFV towards a unified framework for Autonomic Network Management and Control. Thereby, the goal is to take the feedback control loops and architectures from the area of Autonomic Networking (e.g. the ETSI AFI GANA architecture [MCR+16]) and to migrate and extend them by mechanisms and architectural artifacts from the domains of NFV and SDN. Furthermore, [CMS+14] strives to identify synergies with other cross-domain research areas, such as Model-Driven-Engineering (MDE) for network controllers, as well as the utilization of advanced IPv6 mechanisms (e.g. addressing, routing, ICMPv6 based signaling ...) for the network layer.

A number of interesting surveys on SDN and NFV have been completed the last years giving a nice overview of the sheer amount of research and development activities in this area. [LC15] [MSG+16] [RPR16] are examples of such surveys providing a comprehensice overview. [LC15] elucidates on important research and standardization efforts related to SDN and NFV, e.g. the ETSI NFVI architecture. Furthermore, the relationships between SDN and NFV are presented including a historical view of the evolution of these two technologies and their application to the domain of middleboxes (e.g. NAT boxes). This includes examples for available middlebox solutions - such as ClickOS [MAR+14] [MARH13], DPDK [DPD17], and NetVM [HRW15] - and their development over the years. Furthermore, various applications are reviewed and elaborated with the most notable being: Cloud Computing and Data Centers, Mobile Networks, and Enterprise Networks. [MSG+16] largely summarizes the survey results of the previous NFV and SDN reviews, thereby adding some valuable information on possible business models in the emerging ecosystem, including the possibilities for telecommunication service providers, infrastructure providers (e.g. in the data center context), VNF providers and further. In addition, [MSG+16] reviews the variety of standardization activities linked to NFV, such as IETF/IRTF groups [IET17a] [IET17b]. Finally, [RPR16] provides another exciting survey on the application of NFV and SDN in mobile core networks, thereby combining these approaches with the long-term topic of SDR (Software Defined Radio)[29]. Furthermore, [RPR16] describes the various SDN/NFV/SDR scenarios for a

[28] The SDNFV is once again based on previous authors' work related to network virtualization and resulting in the NetVM virtual server platform for network services.

[29] SDR stands for the idea to implement in software many of the hardware components which are in-place during radio communication. This leads to interesting possibilities such as dynamic spectrum allocation for improved radio frequency sharing.

mobile network, such as C-RAN (Centralized Radio Access Network), V-RAN (Virtual Radio Access Network), H-RAN (Heterogeneous Radio Access Network), and finally S-RAN (SDR Access Network). Moreover, besides the survey aspects, an architecture is proposed that enables the integration of SDN, SDR and NFV by utilizing and switching among different control/management protocols (XMPP, MIP, OpenFlow ...) depending on the specific task. A similar goal is pursued in [SKGR15] where NFV, SDN and SDR are being integrated in a common framework for improved controllability of 4G/5G network infrastructures.

Chapter 3

The Self-Healing Function

This chapter aims at elaborating the self-healing function which is a key property and vision of the proposed architectural framework. Thereby, the elaboration starts with presenting different existing understandings of the term "self-healing function" and proceeds with describing the way self-healing is realized in the scope of the current thesis.

3.1 Existing Understanding of Self-Healing

Self-healing is the key function of the technical framework proposed within this work. Therefore, it is imperative to gain an impression of how self-healing was defined and perceived by various research efforts and initiatives, including standardization and industrial activities.

Self-healing was defined as one of the key aspects of Autonomic Computing as introduced by the belonging IBM white paper [ibm05] that pushed for various research efforts related to autonomics. This research includes aspects of traditional IT management as well as Network Management of Internet and telecom type of networks. As previously elaborated, the core concept of this white paper is given by the MAPE (Monitor→Analyze→Plan→Execute) control loop that constitutes the conceptual framework for implementing autonomic behaviors in that context. Correspondingly, the IBM autonomics white paper defines self-healing as *"To discover, diagnose, and act to prevent disruptions."* [ibm05], which is to be realized through corresponding MAPE control loops. Drilling down beyond self-healing, [ibm05] defines the property of self-protection of a system. This is formulated as *"To anticipate, detect, identify, and protect against threats"*. This property relates mainly to security issues regarding a network/system. Indeed, the framework proposed in the current thesis is also meant to deal with security problems, which can be addressed by agent based reactive and proactive mechanisms, i.e. the proposed framework can embed various techniques and flows for Intrusion Detection or Intrusion Prevention. This means that the proposed framework would offer the means to detect in a distributed manner anomalies that might indicate a security attack, would correlate these symptoms and initiate a response as to protect the system/network against the upcoming attack (e.g. DDoS). To tie up the link between *self-healing* and *self-protection* as defined in [ibm05], the distributed framework proposed in this thesis fulfills the *self-healing* as well as the *self-protection* definition from the IBM white paper [ibm05] on Autonomic Computing.

Another basic research publication regarding Autonomic Computing - "The vision of autonomic computing" by Kephart et. al. [KC03] - refers to self-healing as follows: *"Autonomic comput-*

ing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware, perhaps through a regression tester" [KC03] and further states that an autonomic system operates and *"... automatically detects, diagnoses, and repairs localized software and hardware problems."* [KC03]. Similarly as in the case of the IBM white paper, the *self-protection* definition in [KC03] is very close to the *self-healing* one but with focus on security issues, i.e. self-protection aims at mainly resolving security related attacks. This close relation is further pursued in the current work given that the proposed framework for distributed self-healing is also meant to deal with particular security problems, which are resolvable by means of software agents running inside the nodes of a networked system.

Moving on to another notable research and standardization effort related to self-healing: 3GPP has investigated and pushed towards standardization of different self-healing concepts in the scope of SON (Self-Organizing Network). According to 3GPP, self-healing aspects are related to *"Features for automatic detection and removal of failures and automatic adjustment of parameters"* [3GP16a].

Thereby, the self-healing behaviors are captured in the form of self-healing concepts, use cases and requirements in the belonging ETSI standards [3GP16b]. Indeed, each single self-healing behavior is described as a scenario (use case), including the involved components (e.g. base stations, E-UTRAN/EPC components, and logical functions) as well as the steps required to restore the 3GPP access network functionality. To achieve this, a strict definition of self-healing is given which is hierarchically built based on the definition of a *Trigger Condition of Self-Healing* (TCoSH), of a *Self-healing Process*, and finally of the *Self-healing Function* itself. The TCoSH is defined as follows [3GP16b]: *"Trigger Condition of Self-Healing (TCoSH): it is the condition which is used to judge whether a Self-healing Process needs to be started. This condition could be an alarm or the detection of a fault."* [3GP16b]. Based on the TCoSH, a definition of a Self-healing Process is given [3GP16b] by *"Self-healing Process: When a TCoSH is reached, particular action(s) will be triggered to solve or mitigate the particular fault."* [3GP16b], which in turn facilitates the definition of the Self-healing Function that is of particular interest for this work: *"Self-healing Function: a Self -healing Function is to monitor a particular TCoSH and then, if necessary, to trigger a Self-healing Process to solve or mitigate the particular fault."* [3GP16b]. The goal of this definition can be understood as to continuously monitor the condition of the access network and to intelligently react in the face of challenges as to mitigate and/or resolve problems.

Focusing on traditional/legacy networking technologies, self-healing networks were a subject of research and standardization especially with respect to self-healing rings as implemented in FDDI [AJ94], SDH [ITU07], SONET [TEL09], and IEEE 802.5 Token Ring [LS98]. These networks build on a ring infrastructure that provides the possibility to quickly re-route traffic over the alternative ring path in case of a node or link failure. In addition, (G)MPLS network technology [Man13] provides the possibility to setup various alternative paths for different types of traffic (e.g. different classes), such that traffic can be quickly pushed to another route in case of link/nodes failures or QoS degradation in general. Hence, the understanding in that context is again boiling down to the fast recovery of the network services and packet forwarding paths in the face of failures.

C.J. Green discusses on protocols for a self-healing network in his research [Gre95]. In that scope, self-healing is referred to as to *"Provide fault tolerance, with the ability to detect congestion and network faults rapidly and to implement alternate routes without delays for recomputation or hold-down."* [Gre95]. The focus of this early work is set on protocols on the network and transport layer, thereby pointing to congestion, network faults (mainly link and node failures) as well as to the re-

quired rerouting as the key properties of a self-healing network. The framework presented in the current thesis extends this definition by addressing various types of faults - e.g. link/node failures, configuration faults, emerging situations due to failed automations and others described in section 5.3 - in a cross-layered manner spanning the TCP/IP stack.

On the applications layer of the TCP/IP stack, a definition of self-healability is provided within the reference model of the S-Cube (Software, Services and Systems) network of excellence [SCU16], which aimed at establishing a research community for the topic of software services based on Internet technology. The definition is provided in a belonging project deliverable [Del08] and defines self-healibility as follows: *"Self-Healability is the property that enables a system to perceive that it is not operating correctly and, without human intervention, make the necessary adjustments to restore itself to normality"* [Del08]. This prescribes a clear need for a system to reduce (or even) remove the human involvement in the process of restoring normal operation in the face of challenging situations.

In his article "Self-healing Software" [Sah07], Goutam Kumar Saha discusses self-healing as follows: *"Software which is capable of detecting and reacting to its malfunctions, is called self-healing software. Such software system has the ability to examine its failures and to take appropriate corrections. Self-Healing system must have knowledge about its expected behavior in order to examine whether its actual behavior deviates from its expected behavior in relation of the environment. Self-Healing system in general has similar objectives to the area of dependable computing system. Techniques involved in self-healing are similar to those of dependable computing techniques but all dependable computing research areas are not self-healing."* [Sah07]. The especially interesting aspect of this definition is given by naming the link between self-healing and the area of traditional dependable computing. Indeed, techniques and reaction schemes from the area of dependable computing should be taken into account in the course of designing a self-healing system/software. However, the above discussion/citation refers mainly to inbuilt features within a software program - e.g. particular type of exception handling, whilst the research in this thesis aims to establish an agent based framework, which should be aware of the expected normal behavior of a particular distributed communication based system, and should strive to maintain it and optimize its operation at the same time.

Based on the above understandings and definitions of self-healing, the next subsection elaborates on how the self- healing function is realized within this work.

3.2 Self-Healing Function as Realized Within the Current Thesis

Given the reviewed understandings of self-healing, one can observe that they all involve two key aspects, which are given by a proactive and a reactive self-healing behavior of the system in question. That is, a self- healing system should be able to reactively deal with occurring challenges, as well as to proactively anticipate and avoid impending problems. In this line of thought, the current work builds on an abstract definition which meets the above key needs and was set in a previous work of the author [TC10b] based on the fundamental definitions in [ibm05]. Hence, for the purpose of this work self-healing is understood as follows:

To detect incidents such as adverse conditions, faults, errors, failures; diagnose, and act to prevent or solve disruptions.

That is, on one hand a networked system equipped with self-healing mechanisms - given by the emerging agent based framework - should try to automatically remediate future problems, whilst

on the other hand it should detect, diagnose and remove faults, given that the detected/diagnosed faults can be automatically resolved, or have their impact reduced to a minimum, by means of a software-agent based framework.

The current thesis proposes to realize self-healing for a distributed networked system based on the deployment of software agents in the network nodes. These software agents collaborate and implement self-healing in a distributed manner thereby fulfilling the above requirements for proactively and reactively handling network/system challenges. Thus, the proposed framework implements a distributed control loop, in which every node can implement reactive/ proactive schemes based on information regarding faults, errors, failures and alarms, which is shared between the network devices. Indeed, the current thesis devises a set of components which should be in place within each node of a self-healable network/system. These components facilitate each device in the network

1. to monitor for anomalies and symptoms of faulty conditions.

Subsequently, the detected symptoms are pushed to a local agent that is generally responsible for:

2. disseminating the information among the involved network nodes.

Given that some incidents/symptoms where either locally detected or received from the network - based on step (2) from above, these incidents are pushed to other software agents within a network node, which are in turn responsible for:

3. issuing a fast reaction to the detected symptoms in order to mitigate their immediate impact
4. analyzing the symptoms w.r.t. to future potential fault activations and triggering a corresponding action, and in parallel
5. analyzing the overall set of incidents (gathered across the network) in order to identify the root cause(s) for the observed problems and
6. removing root cause(s) in the long-term operation of the network.

In this line of thought, the most important ingredient is given by the fact that steps (3), (4), (5) and (6) are executed not only for the locally detected, but for the overall set of symptoms and incidents which are observed across the network. This is enabled by the mechanisms for incident sharing among the involved devices, which aim at delivering the detected information among the nodes in question. This incident exchange operates whilst the network/system is facing challenges and/or faulty conditions, and hence should be implemented in a way as to increase the probability for a successful message delivery, given the problems faced by the network/system as a whole. More details are given in the belonging chapters relating to incident information exchange.

Whilst realizing the above behaviors the components aim at synchronizing their actions towards achieving optimal operation and striving to realize distributed self-healing in a way that provides stable networking services and better QoS to the end user. These self-optimization aspects as well as other self-optimization features of the proposed framework are elaborated in the next chapter.

Chapter 4

The Self-Optimization Function

Having presented the self-healing function, as understood within this thesis, the next step is to introduce the self- optimization function, which is the second key feature of the proposed self-healing framework. As in the previous chapter, the presentation starts with a discussion on existing definitions and understandings of the term self- optimization. Thereafter, the realization of the self-optimization function within the proposed self-healing framework is elaborated.

4.1 Existing Understanding of Self-Optimization

Self-optimization is considered one of the key features of an autonomic system as introduced within the fundamental IBM white paper [ibm05] on autonomics. According to [ibm05], self-optimization should be realized through the use of MAPE (Monitor→Analyze→Plan→ Execute) control loop structures. Such MAPE control loops constitute the means to meet the following definition of self- optimization: *"To tune resources and balance workloads to maximize the use of information technology resources."* [ibm05]. Obviously this definition focuses on the optimal utilization of IT resources through the adjusting of corresponding parameters, which is also a vision followed within the design of the emerging distributed framework for self-healing.

Another fundamental work in the area of autonomic computing that discusses on aspects of self-optimization is given by [KC03]. This work refers to complex middleware or database systems that *"may have hundreds of tunable parameters that must be set correctly for the system to perform optimally"* [KC03] and argues that only *"few people know how to tune them"* [KC03]. In that scope, [KC03] defines the role of autonomic systems with respect to self-optimization: *"Autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost"* [KC03]. The current thesis clearly adopts this vision by devising a framework that tries to execute self-healing as to not only repair basic network/system services, but also aims at restoring the system to an optimal configuration, thereby avoiding inconsistencies and instabilities in the required and initiated self-healing reactions. In addition, the proposed self-healing framework aims at optimizing its own operations, e.g. when it comes to the monitoring overhead induced in the involved network nodes.

The emergence of 4G mobile networks was accompanied by a strong push towards the self - optimization of the access network segment, which resulted in standards such as SON. Different companies - e.g. vendors and consulting services - were engaging in that scope and were refining the practical understanding of self-optimization in the light of 4G access network technology.

Referring to research efforts in the scope of 4G, an interesting work [HZZ+10] states that self-optimization includes approaches "such as CCO" (Coverage and Capacity Optimization) ... "MRO" (Mobility Robustness Optimization) ... ", mobility load balancing, and interference control", as standardized in LTE-Advanced. Thereby, this work elaborates on the introduction of MAPE like control loops for self-optimization within the SON access network with the aim of reducing OPEX and CAPEX.

Similarly as in the above research, the keywords and acronyms definition of 3GPP [3GP16a] relates to self-optimization for 4G access networks as including *"optimization of coverage, capacity, handover and interference"*. However, the 4G approach is pretty tight to particular functions and behaviors for self-optimization of the radio access technology. On the contrary, the current thesis goes for an abstract approach that is generally applied for whatever type of self-healing actions and behaviors are configured/realized by the proposed distributed framework. This can easily incorporate, but is not limited to, radio access network related self-healing and self-optimization features.

4.2 Self-Optimization Function as Realized Within the Current Thesis

The following paragraphs aim at elaborating the role of the self-optimization function within the current thesis, i.e. in the course of the proposed architectural framework for distributed self-healing.

Firstly, a definition is required which would introduce and identify the place of self-optimization during the envisioned self-healing process. Based on the above understandings and definitions, self-optimization is regarded as follows in the course of this work:

The proposed framework for self-healing aims at recovering the network/system functionality in a way, and to a state, which is optimal for the network/system in question, such that it can continue delivering its services from this state.

In addition, the proposed framework for self-healing should continuously seek to improve its own operation in terms of recovery strategies and resource utilization (e.g. with respect to overhead) within the network nodes.

The above understanding is based to a large extend on the definitions given in [ibm05] and [KC03], and cited above. These definitions are combined and adapted to the context of the emerging framework for distributed self-healing, where the self-optimization refers to the need to improve the operation of the framework itself, while implementing self-healing behaviors within the network nodes. In this line of thought, the self-optimization features of the emerging framework are listed below:

Monitoring Job Parameters: The proposed framework for distributed self-healing relies heavily on monitoring functions for the detection of symptoms of faulty conditions. These monitoring functions induce some overhead in the devices and the network as a whole, e.g. memory usage or bandwidth usage/overhead (in case of some active or passive network monitoring techniques). Hence, in order to reduce this overhead, it is imperative to regulate and self-optimize with respect to various parameters of the belonging monitoring jobs such as sampling rate, aggregation function parameters - in cases where several measured values are aggregated before further utilization, or number of samples to keep track of in a node local database - this would need to be regulated

in order to manage the amount of monitoring data stored within a node.

Fault-Propagation Model: The Fault-Propagation Model[1] deployed across the network nodes constitutes the key model that steers the Fault-Isolation process and determines the reactions of the emerging self-healing framework. Naturally, a Fault-Propagation Model would be based on some probabilistic framework (e.g. Bayesian Networks), which would isolate potential faults with a certain probability. The self-healing framework would in turn check whether these faults are indeed present and would also provide the possibility to implement mechanisms which could automatically adapt the Fault-Propagation Model instances in case the diagnosis results have turned out to be incorrect. The adaptation would naturally consist of adjusting the probabilities in the scope of the belonging probabilistic model.

Context-aware Self-healing Actions: The experiments that were conducted in the course of this thesis have clearly shown the need for adapting the actions issued by the self-healing framework, depending on particular context such as the history of executed actions. To justify that: for instance it was experienced that restarting a particular type of Network Interface Card (as a self-healing action) has led to a situation where the corresponding driver was not functioning properly and the whole NIC or even node needed a restart, in order to resort from this erroneous state. Hence, it is required to understand such situations and plug-in such context -aware self-healing behaviors using the hooks provided by the proposed framework for distributed self-healing.

Selection of Optimal Tentative Actions: Various agents of the proposed framework, operating in different nodes across the network, would plan some tentative actions in order to either realize self-healing by executing some corrective actions or to regulate some aspects of the framework's operation (e.g. monitoring job parameters). These tentative actions require to be synchronized among each other such that the right subset is selected, in order to recover the system to an optimal state, as well as to converge and improve with respect to the best self-optimization actions for the distributed framework itself. As a means to achieve this, a self-optimization model is put in place that allows for selecting the optimal subset of tentative actions issued by the involved agents (of the emerging self-healing framework) after having executed their belonging control loops.

Avoiding Contradictory Tentative Actions: The self-optimization function of the proposed framework inherently takes care of avoiding contradicting tentative actions, which are either related to self-healing or to optimizing the operation of the distributed self-healing framework. Thereby, the contradictions are avoided based on the mechanisms for self-optimization provided by the corresponding agent (later denoted as Self-Optimization Agent) of the framework within each node. This means that the optimization of the state, to which the self-healing operations aim to recover the network/system, turns to be the vehicle to achieve the avoidance of contradictory self-healing measures across the distributed system or network. Thereby, actions are allowed or disallowed based on a model for the self-optimization of tentative actions as proposed later on in this thesis.

Having described the self-optimization features/function of the emerging framework for distributed self-healing, the next chapters proceed with designing the node and network level components towards the realization of distributed self- healing with self-optimization aspects. This includes the specification of the components, their interfaces and interactions, as well as the algorithms which are required to implement scalable and efficient distributed self-healing with self-optimization features.

[1] The model based on which network challenges are analyzed and the root causes identified.

Chapter 5

Research Requirements Analysis

In this chapter[1], a list of requirements is derived that determines various aspects, components and mechanisms of the emerging *FDH*(*Framework for Distributed Self-Healing*). While it is a challenging task to capture all possible requirements, the list here was gradually enhanced in the course of various activities - including large scale research projects on EU level - and constitutes an exhaustive set of aspects reflecting the plethora of issues related to self-healing. The requirements that are presented in this chapter will drive, within the rest of this thesis, the definition of various components and mechanisms addressing faults/errors/failures/alarms on node and network level.

First, a set of functional requirements for the FDH framework are given after a short discussion, in which each of these requirements is motivated. Secondly, a number of non-functional requirements are presented which concern various aspects of the operation of FDH. Such aspects are given by the security of the framework, its scalability and overhead produced in the network nodes. Finally, the different types of faulty conditions are outlined, which the emerging framework for self-healing is meant to handle. This poses on one hand a set of requirements to be met by the design of the FDH architectural framework, whilst on the other hand the list of the targeted faulty conditions defines the operational scope of FDH.

5.1 Functional Requirements

The list of requirements derived in this section determines various aspects, components and mechanisms of the emerging framework for distributed self-healing. As previously mentioned, while capturing all possible requirements towards such a framework constitutes an extremely challenging task, the current list was gradually updated and improved as a result of an extensive literature review, as well as various discussions in the course of European projects, e.g. ANA [BJT+10] [ANA16] and EFIPSANS [efi16a] [efi16b], plus discussions around emerging standardization efforts in the area [SWC+12] [SWC+] [ETS13] [CWM+13]. Thus, the coming list can be considered an exhaustive list reflecting the plethora of issues related to agent based self-healing in distributed networked environments.

[1] The current chapter is based on initial considerations published in [TS14a].

5.1.1 General Functional Requirements

First of all, the FDH must provide interfaces and components addressing the tasks of Fault/Incident-Detection, Incident/Alarm-Dissemination across the network, Fault-Isolation, and Fault-Removal. That is required, in order to enable the FDH to take care of faulty conditions across the networked system in a distributed manner. Therefore, the presence of erroneous states needs to be first detected and consequently disseminated to all FDH running nodes, such that the overall set of FDH instances can react adequately. This reaction is meant to ultimately lead to the removal of the root causes (Fault-Removal) for the observed symptoms. Thus, the root causes must have been identified by corresponding components for Fault-Isolation, running within the scope of the FDH instances in the network nodes. These considerations lead to the following concrete requirement:

Req 1: The proposed architecture must define components and interfaces that target issues related to the processes of Fault-Detection, Incident/Alarm-Dissemination across the network, Fault-Isolation, Fault-Removal, and Fault-Mitigation.

As mentioned above, the goal is to design a framework which operates in a distributed manner with its components embedded within the network nodes. By devising the framework to operate in a distributed manner, the traditional principles of network management are bridged, where network management systems constitute centralized instances (e.g. SNMP type of network management architectures). Such a distributed architecture for self-healing allows to reduce the overhead of gathering related information at a central place, and facilitates the fast realization of resilient mechanisms as close as possible to the network nodes. This results in the following requirement.

Req 2: The proposed framework should remove/mitigate problems operating as close as possible to the device, thereby working in a distributed manner without (fully) relying on a centralized instance.

In the course of its distributed operation, the FDH components would require to access and modify various parameters and settings within a network node as well as the network as a whole. For example, such exclusive access might be required in the course of Fault-Removal, i.e. whilst removing the root cause of the observed set of symptoms, as well as during monitoring and verifying the different steps of the FDH control loops' executions. Therefore, the FDH entities within the network nodes should be given exclusive access to every functional entity and resource inside a device in order to manage it with respect to alarms and incidents, and thus enable the process of distributed self- healing.

Req 3: The components of the architectural framework must be given exclusive access to all functional entities and resources inside a node in order to manage them with respect to incidents and alarms towards the realization of scalable and efficient distributed self-healing.

5.1.2 Monitoring Requirements

Within the emerging FDH, a monitoring component responsible for Fault-Detection as well as for gathering general monitoring information is introduced in every networking device. This means

that monitoring entities in multiple devices collaboratively monitor and process information regarding the involved network nodes. This implies a number of requirements, such as the need for the monitoring agents (across the FDH nodes) to collaboratively detect faults and monitor various key performance indicators, both on network level and inside a host/router. These considerations result in the following requirements.

Req 4: Distributed monitoring realized by collaborative behaviors between different FDH monitoring components is an essential aspect to be addressed by the monitoring parts of the emerging self-healing framework.

Req 5: The Fault-Detection/monitoring aspects covered by the FDH monitoring components, which are dispersed across the network, are given by the need to detect faults based on diverse parameters within each single hosting node. In addition, the FDH monitoring components inside a device should be able to monitor and detect faulty conditions based on diverse aspects related to traffic which passes through the node.

Within the scope of the FDH, it is further essential that the monitoring components can serve on-demand requests submitted by the various agents of the emerging framework. Thereby, the requests should trigger the instrumentation of monitoring tools which gather information about (multiple) key performance indicators. Furthermore, the monitoring components of the FDH should hide the complexity of orchestrating the required monitoring tools, whilst obtaining the belonging KPI values and searching to detect erroneous states in the network. Following requirements arise due to the above aspects.

Req 6: The FDH monitoring components inside a node should be able to accept and serve monitoring requests submitted by another FDH agent or on demand by the network administrator. That is, other autonomic agents should be able to request the FDH monitoring agent to start/instrument different type of monitoring techniques, extract and analyze belonging information, and finally recognize conditions (e.g. faults or any other situations worth disseminating) which are to be described and conveyed back to the requesting entity.

Req 7: The FDH monitoring components inside a device should hide from a requesting agent (in the sense of the previous requirement) the complexity of orchestrating monitoring tools. That means, the requesting agent, should get notified of the requested monitoring information or incidents without being involved in the specifics of the monitoring process/ activities i.e. specific monitoring tools, protocols, information sources etc.

Moreover, the FDH needs to include a monitoring database in each node. This database should keep important KPI measurements which should have been preprocessed/aggregated, in order to reduce the total amount of measurements to a manageable and meaningful number.

Req 8: Each FDH node must keep a log (database) of the KPIs values monitored over a particular time period.

Req 9: Aggregation of monitoring data should be part of the process of analyzing the data towards detecting erroneous states and/or providing important KPI measurements for the requesting agents.

The extensibility of the FDH monitoring platform is very important, in order to implement the efficient monitoring for different KPIs and to facilitate Fault-Detection for different types of faults according to the specifics of the underlying network/system. On one hand, the extensibility relates to extending the monitoring features of the platform, and on the other hand to providing hooks for realizing self-optimization within the FDH monitoring components. That is, the latter type of hooks would allow for dynamically instrumenting policies concerning aspects such as 1) admission control for monitoring requests, 2) notification rates for requesting agents, and 3) adaptation of running monitoring services towards maximizing the effectiveness and minimizing the overhead.

Req 10: The monitoring components should be easily extensible by different monitoring features/plug-ins that allow acquiring information about diverse metrics and key performance indicators towards the detection of faulty conditions. Examples of relevant metrics and KPIs are given by: 1) one-way delay, 2) round-trip delay, 3) available bandwidth towards some recipients, 4) CPU utilization (e.g. along a path), 5) utilization of the queues in the routers along a path, 6) number of dropped packets on different layers (e.g. TX and RX of a NIC) etc.

Req 11: The FDH monitoring components inside a node should provide hooks for supplying different types of policies and implementing different adaptive behaviors and mechanisms. Such behaviors and mechanisms might be related to admission control with respect to requests for establishing monitoring services, adaptation of monitoring services (e.g. sampling rate), or escalation policies determining which values should be reported to the requesting agent.

The above requirements determine design of the FDH monitoring functions. Having detected an incident, the FDH needs to disseminate the information to relevant FDH agents across the network, which is the scope of next subsection.

5.1.3 Incident/Information Sharing Requirements

Once a faulty condition has been detected based on the above requirements, or an incident has been recovered and a recovery notification is pending, the corresponding node components of FDH are expected to step up and disseminate the information in question to all FDH instances across the network. This implies that FDH should provide hooks to trigger incident information sharing and additionally should put in place the components and mechanisms facilitating the exchange of vital self-healing information (e.g. incident description and incident recovery notifications). The following paragraphs elucidate on and formulate requirements to these aspects.

An important aspect to be considered is that of the interfaces which are available for incident/alarm information sharing. That is, it should be possible to access the FDH over a well-defined interface and submit information about any network/system malfunctioning. In this way, developers can implement software as to share information regarding erroneous states over the FDH, and to support the framework in the course of achieving network resilience through agent based self-healing. For example, a Java or C/C++ based service/application might be developed in a way that in case of exceptions related to network resource unavailability, the situation is described and the belonging information pushed to the FDH framework in order to support the distributed self-healing. The same interfaces are to be used for alarm and incident sharing between the FDH

components across the network in question, e.g. Fault-Detection entities operating as monitoring plug-ins in the scope of FDH. Hence, the following requirement emerges.

Req 12: The architecture must provide components and interfaces to enable the collaborative sharing of alarm/incident information inside a device and across the network.

Given the fact that in case of an erroneous state a large amount of incidents might emerge, the FDH incident/information sharing mechanisms should be able to reduce the dissemination activities to the most necessary ones. The task of reducing the amount of outage related information, which is conveyed to the centralized Network Management System, is known under the term *alarm suppression* [JWW94] in traditional network management. Thereby, different techniques are applied in order to suppress irrelevant information/alarms, or information/alarms of minor importance, along the hierarchy of devices and management systems underneath the centralized Network Operations Center. In the case of FDH, a strong tool to reduce the amount of disseminated incidents/information is constituted by the possibility to send particular messages only to those nodes, which can adequately include the information in the belonging incident handling processes (e.g. Fault-Isolation, Fault-Removal and Fault-Mitigation).

Req 13: The issue of *"alarm suppression"* must be addressed in the course of the process of alarm/incident dissemination. That is, alarm/incident descriptions should not be flooded to all, but should only get conveyed to selected nodes, which have to be informed of, and/or can benefit from the corresponding information.

Req 14: The employed dissemination techniques should be able to deliver incident notifications to pre-defined groups of nodes, i.e. they must incorporate multicast mechanisms. Furthermore, the alarm/incident dissemination mechanisms should also have the ability to notify entities residing on network nodes or end systems, which are not belonging to any multicast group.

In general, the inter-node alarm/incident sharing mechanisms would be operating in the face of challenging conditions whilst the network as a whole is trying to survive. Hence, these mechanisms would need to consider the increased probability of failing to deliver the required messaged and should correspondingly try to remediate potential difficulties. This means that the alarm/incident information sharing - among the involved FDH nodes - should depend on the situation in the network, as well as on the importance/severity of the alarm/incident to disseminate. Hence, following requirements emerge.

Req 15: An appropriate alarm/incident dissemination strategy (among the FDH nodes) must be selected according to: 1) the severity of the information to convey, and 2) the situation in the local node and the network.

Req 16: In the context of employing unreliable communication protocols - for example (UDP on top of) pure IPv6 multicast or unicast, the dissemination techniques should aim at increasing the chances of the alarm/incident information to get conveyed to the intended recipients.

5.1.4 Incident Handling Requirements

Having covered the aspects of monitoring and self-healing related information dissemination, the next step is constituted by defining requirements towards the way incidents are handled within

the FDH nodes in a distributed manner. These requirements prepare the path towards efficient distributed self-healing with self-optimization features.

The various symptoms, which are visible during an erroneous state, should be optimally analyzed, in order to identify the belonging root cause(s) with the goal of eventually removing it/them. This requires the process of Fault-Isolation to be supplied with models that facilitate it to quickly and efficiently identify faults based on the observations made across the network. In that line of thought, for the purpose of Fault-Isolation, the causality relations (fault-propagation models) among the observable symptoms (failures and alarms) and the internal propagation of events in terms of root causes (faults) and errors should be taken into account. This would result in a Fault-Propagation Model, which will be used for Fault-Isolation within the network devices.

Req 17: The framework must consider the causality relations (fault-propagation patterns) regarding root causes (faults) and their propagation as one or many errors until eventually resulting in observable symptoms (failures and alarms). These causality relations should be provided as a Fault-Propagation Model based on which the process of Fault-Isolation would be realized in the network nodes.

Once a Fault-Isolation result has been obtained based on the above Fault-Propagation Model, this result must be verified. On one hand this requirement aims at avoiding consecutive actions based on wrong information. On the other hand, by checking the quality of the obtained Fault-Isolation results, the underlying Fault-Propagation Model can be improved in the long-term operation of the network. These thoughts result in the following FDH requirement.

Req 18: The results of the Fault-Isolation process should be verified within the network nodes.

In the context of Fault-Removal, it is imperative that the framework takes into account the dynamics of the node and the network with respect to the history of executed management actions. For example, in some cases if a Network Interface Card (NIC) has been restarted a number of times, the driver might not able to reinitialize after a next restart, and hence only the rebooting of a node can be used as a last resort to restoring full functionality. Hence, in such cases the Fault-Removal functions of FDH should be able to understand the situation and undertake the right action as to restore full functionality.

Req 19: When executing Fault-Removal actions, the dynamics of the software and hardware components with respect to the history of executed management actions should be taken into account.

The FDH framework embeds different processes and belonging control loops (e.g. Fault-Removal). These would mainly concern the handling of faulty conditions, but also the optimization of the framework's operations, e.g. the operational fine tuning of the monitoring functions. Hence, all these control loops and their multiple instances - for example parallel running Fault-Removal processes - must be synchronized and their tentative actions optimized, in order to achieve an efficient FDH operation and increased QoS of the underlying network in question. Therefore, in order to ensure the stability with respect to concurrently/parallel running FDH processes and control loops (e.g. in a multi-threading environment), the FDH needs to guarantee that the overall set of actions to be executed are synchronized towards a common goal of improving the overall fitness of the network. Hence, a component is required that synchronizes, in an appropriate way, the actions issued by diverse concurrent FDH processes.

Req 20: The emerging architectural framework needs to ensure that the actions undertaken by parallel FDH processes do not contradict each other and that the entire decision making process effectively contributes to the overall fitness of the network.

In addition, after executing a Fault-Removal action on a resource or managed functional entity, the success of the action must be checked, in order to verify that the functionalities in question were properly recovered. This would make the FDH framework aware of the success of its activities, and will contribute to the stability of the architecture by providing means to assess the quality and trends in the undertaken agent based managerial actions.

Req 21: The effectiveness of the undertaken Fault-Removal management actions must be assessed.

Last but not least, the FDH instances in the network nodes across the network should be able to self-assess and judge on whether they are successfully coping with the challenging conditions the network is exposed to. Indeed, in situations where FDH is not able to remove faults and improve the network operation, the issues should be escalated to the network operations personnel. Thereby, all the relevant information should be forwarded to a centralized Network Management System, such that the network administrators can analyze it beyond the capabilities of FDH. The goal of the following requirement is to ensure that FDH keeps the human in the loop and does not perform useless operations or even harm the network, in case it is incapable to address the current challenges.

Req 22: The framework should be able to reason whether it is able to handle the overall set of faulty conditions in the network, and in case it is not, to escalate the challenging situation to the network operation personnel.

The functional requirements that were presented in this section will drive, within the next sections, the definition of various components and mechanisms addressing faults/errors/failures/alarms on node and network level.

5.2 Non-Functional Requirements

Non-functional requirements relate to the characteristics of the system's operation in general. In this line of thought, functional requirements specify the functionality of a system, whilst the non-functional requirements describe the way these functionalities are executed and the system properties, which accompany its operation and contribution to an ecosystem. There are various standards which specify the type of non-functional requirements to be covered while designing and implementing a system [ISO01] [ISO10] [SB13]. Given the fact, that the current work focuses on the research aspects relating to the emerging framework for distributed self-healing, non-functional requirements looking at FDH as a product are omitted, e.g. usability, portability, and look and feel. Instead the focus is set to requirements which are vital to be addressed by the research approach at the pre-product/prototype phase - e.g. security, scalability, dependability, stability, efficiency and overhead.

The first non-functional aspect to pay attention at is Security. In that scope, the FDH should be designed in a way as to be secure against threats and manipulations. Especially, the traditional

dimensions of security -Confidentiality, Integrity, and Availability (CIA) [LSG16] [GvSO01] - should be taken into account, and the FDH should be able to employ mechanisms, which address the above aspects. For example, the signaling mechanisms between the FDH instances should be secured against manipulations, such that the FDH cannot be highjacked by hackers and turned against the integrity of the underlying system/network. Furthermore, the availability of the FDH instances, even in the face of challenges to the hosting network elements, should be considered and addressed in the FDH designs. These considerations lead to the following requirement.

Req 23: The FDH framework should be designed in a way, such that vital security aspects are covered and the framework instances do not open vulnerabilities that might be exploited by attackers. Especially, the CIA aspects of security should be considered. Thereby, it is important to ensure integrity and confidentiality with respect to the signaling information flowing among FDH instances and components, as well as to take measures to sustain FDH availability/services in the case of challenging situations[2].

Secondly, the aspect of scalability needs to be elaborated on. Scalability is generally perceived as the capability of a system to process growing number of inputs within a reasonable time/computational and space (e.g. memory) limits. Hence, it is imperative for the FDH to be able to process growing numbers of events within acceptable limits in regard to computational and space (memory) complexity. This aspect relates mainly to the key mechanisms and algorithms, which will be employed within the FDH instances.

Req 24: The distributed FDH control loops should be realized through scalable algorithms and mechanisms, thereby allowing the FDH to process reasonably growing numbers of faults, errors, failures, and alarms.

The issues of efficiency and overhead are closely related to the aspect of scalability. Indeed, it is imperative that the FDH requires minimal CPU utilization and memory consumption for its operation, even in the face of large numbers of incidents which need to be processed. In addition, the execution times for the control loops should also stay minimal with a growing number of incidents and size of the employed models (e.g. Fault-Propagation Model, Policy Models etc.). The latter execution time aspect is very important, as to enable the FDH to quickly remove or mitigate erroneous states and to help the operators provide an acceptable QoS even in the face of network challenges.

Req 25: The components and phases of the FDH control loops should employ efficient algorithms thereby reducing the FDH overhead in terms of memory consumption and CPU utilization within a network node. In addition, the execution times of the FDH control loops should remain reasonably minimal, whilst the size of the employed models (e.g. Fault-Propagation Model) and the number of events to process in parallel might be growing.

With respect to dependability and stability: the FDH design should consider these aspects from different angles. First of all, the FDH prototype should prove that such a framework can operate in a stable way in the long-term operation of the network, thereby removing/mitigating faults

[2]The current requirement is not thoroughly addressed in the scope of the current thesis, since the focus is mainly set on capturing the overall requirements, devising the concepts/algorithms, and proving the applicability and benefits, as well as the scalability of the emerging framework for self-healing.

and supporting the network operations personnel to sustain a required QoS level. Furthermore, it should be possible for the FDH to continue its operation even in the face of serious challenges including the outage of hardware and software components, as well as in case that even parts of the FDH itself have crashed. Moreover, the FDH design should take into account that the resulting actions, which are undertaken as measures to reduce the effect of or to completely remove faults, might after all result in contradictions between single actions and even harm the overall health of the network/system. Thus, the FDH specification should take into account the issue of contradicting control loops and provide means to avoid such situations.

Req 26: Regarding dependability and stability in relation to FDH, the emerging framework should be able to: 1) operate in a stable manner in the long-term operation of the network/system and support the administrators towards sustaining a required QoS level, 2) continue its operation in the face of network/system challenges even in case when parts of the FDH itself have crashed, and 3) consider the issue of potential contradictions and unwanted oscillations between managerial actions and provide means to avoid such.

Having captured the vital functional and non-functional requirements for the emerging framework for distributed self- healing, the following section outlines the operations' domain of FDH by classifying the different types of problems to be addressed by the framework. In addition, the role of FDH is contrasted and put in relation to the way IT system and network components traditionally address the particular type of problems.

5.3 Classification of Problems

The current section classifies the various types of erroneous states and problems which are to be handled by the distributed FDH architecture. This classification outlines the scope of operation of the emerging architectural framework and constitutes a requirement for it, i.e. FDH should be designed in a way as to be able to handle the listed types of problems in an appropriate way. Furthermore, appendix C shows some concrete examples of network problems, which can be seen as instances of the classes of faulty conditions below.

Whilst presenting the different classes of faulty conditions and problems, the role of FDH based self-healing in relation to the particular type of network challenge is also sketched. Furthermore, the way standard Fault-tolerant Systems, Services and Protocols (in the following denoted as Domain-FSSP) handle the particular type problems is contrasted to the way FDH deals with them (denoted as Domain-FDH). This contrast is important as an attempt to draw a line between the operational domain of FDH and the resilience features existing within existing network protocols, services and applications.

5.3.1 Faults/Errors/ Failures detected at the Network Layer, Link and Physical Layers

Domain-FSSP: Some protocols are designed with intrinsic mechanisms to detect faults/errors/failures and react by executing resilience and recovery mechanisms i.e. fault-tolerance mechanisms and strategies, e.g. protection and restoration schemes in telecommunication networks [TH01], [ITU98], [Aut03]. For example telecommunication protocols such as ATM, MPLS/IP-FR, and SONET/SDH exhibit such resilience and recovery characteristics [ITU98], [Aut03].

Domain-FDH: The Fault-Detection part of FDH needs to not only rely on NMS level[3] alarms generated as symptoms, but also access detected errors/failures and information in general regarding low level conditions/states, which are normally not communicated to the NMS through alarms. There might be various reasons for such information not being visible on the level of NMS, e.g. alarm suppression or simply because "humans" may have thought that some low level incidents need not be communicated or even archived for offline analysis.

Moving on to Fault-Isolation as a next step to FDH based self-healing, Alarm Correlation (and correspondingly Fault-Isolation) is traditionally performed using NMS level alarm information coming from the network elements. For FDH based self-healing, some information about low level detected incidents - often not considered necessary to be communicated to the Network Management System (NMS) - as well as detected threats and attacks may be required in performing extensive Fault-Localization. An important constraint in that context is given by the question on whether it is practically feasible to include all such information in a Fault-Propagation Model of a device or a network of some scope.

Following the Fault-Isolation aspects of FDH, the processes which influence the network as a result of an erroneous state, namely Fault-Mitigation and Fault-Removal, need to be elaborated in the scope of the current type of faulty conditions. Fault-Removal and Fault-Masking mechanisms must infer whether resilience and recovery, i.e. fault-tolerance mechanisms, employed by some protocols upon the detection of an incident have executed successfully or not. This is required, in order to execute a strategy on the global level, to remove or reduce the impact of a fault, in order to ensure system/network integrity, reliability and availability. Fault-Removal should embed information regarding the dependencies between the different networking components (software and hardware) as discussed in [LCZ08] and later in this work. Decisions regarding Fault-Removal will be taken mainly locally. However, if required, a dedicated instance might initiate actions on network level. These actions might be executed in a distributed and collaborative manner among the involved network nodes/systems.

5.3.2 Faults/Errors/Failures detected at the Services and Application Layers

Domain-FSSP: There is a great deal of work achieved and research continues in systems engineering practices for the design of fault-tolerant service components and applications. Multi-Layer resilience strategies (see [TH01], [Aut03]) also include how the service/application layer reactions should be taken into account. Furthermore, the efforts in the area of Content Distribution Networks (CDN) and Cloud Computing have resulted in highly resilient service architectures (e.g. based on cloud elasticity and load balancing), which are able to dynamically reconfigure and successfully address challenging conditions, whilst continuing to provide the requested services.

Domain-FDH: Fault-Detection in the services/applications layer follow similar requirements as discussed in the case of Fault-Detection at the network layer and below. Namely, the goal of the Fault-Detection mechanisms instrumented by FDH is to obtain a rich set of information regarding operational issues related to services and applications, beyond the scope of the alarms reported to the standard OSS (Operations Support System). As the operator's desire to see automated service monitoring (refer to [WL09]), it means FDH related mechanisms must perform service monitoring, e.g. service probing to detect problems.

With respect to Fault-Isolation, Fault-Propagation Models are required which describe the different potential problems and fault propagation chains across the various communication layers up

[3] Network Management System level

to the services and applications layer.

The same is valid for Fault-Removal/Mitigation policies related to the service/application layer, which enable the reaction to root causes of erroneous states (obtained based on the above mentioned Fault-Propagation Models) as well as to the symptoms manifesting erroneous states. Thereby, the aspects of cross-layer fault propagation (e.g. between network and service layer) should be considered whilst analyzing the symptoms of erroneous states, and in the course of removing or mitigating those.

5.3.3 Traditional Network and Systems Management Alarms

Domain-FSSP: At this level, some resilience and recovery functions process alarms and react upon them, while at the same time generating other types of alarms to the Management Systems or OSSs.

Domain-FDH: Alarms are used primarily as symptoms used for triggering Fault Diagnosis/Localization. Alarm "severity" indications are vital input to Fault-Removal and Fault-Mitigation functions.

5.3.4 Human Errors during Network and Service Management Phase

Domain-FSSP: Such Human Errors are not handled at this level, but rather in the FDH domain.

Domain-FDH: In [Whi08], an insight is provided as to what sort of faults may be created during the configuration of the network by humans, e.g. erroneous configuration data provided as input to systems. Moreover, [Whi08] and [STP+15] discuss automation techniques that relieve the operator from introducing severe faults into the input supplied to configure network elements. Despite such research and development efforts, the introduction of faults into network configurations cannot be completely avoided. Indeed, configuration faults will be predominantly manifesting in particular situations, which deviate from the normal operation of the network, e.g. whilst the network is trying to cope with some challenges and dynamically reconfigures as to preserve an acceptable QoS. In such situations, the FDH should support the network components by detecting the corresponding symptoms and analyzing the situation, based on a Fault-Propagation model that takes into account known configuration based problems. Finally, the FDH should try to remove the fault by adapting the configuration whilst at the same time reacting to the manifesting symptoms, in order to sustain the network QoS. Hence, the autonomic FDH agents should incorporate - on the base of the corresponding types of reaction policies and models - fault-tolerance techniques (e.g. as described in [Whi08]) or in general Fault-Removal techniques such as Rollback, Compensation, Reconfiguration, Re-initialization, etc. To summarize, the FDH agents should be able to discover a manifesting erroneous configuration and correct it during the operational phase of the network.

5.3.5 Unknown Faults resulting from insufficient Testing of Software

Domain-FSSP: Resilience and recovery mechanisms within the software components can be expected to handle such faults in a very limited scope.

Domain-FDH: Such kind of faults are hard to detect and identify, and would be often classified as "unknown" during Fault-Isolation. Naturally, they would be resolved by upgrading the software components, by applying patches and resorting to new releases [Whi08]. In situations where the

Fault-Isolation process is not able to conclude regarding the root cause for an erroneous state, the faulty condition should be escalated to the network operations personnel (also denoted as "human in the loop"), and all relevant information should be provided as to enable the successful analysis of the situation by the network administrators. Given that the problem is analyzed and concluded as a deficit/ bug of the involved software components, these components should be either updated (if possible at this point of time), or the FDH models/policies should be changed as to accommodate the experienced type of faults, until the improved updated software is available. Provided that the FDH models/policies have been adapted, the FDH should be able to detect the manifestation of situations, for which the aforementioned software weakness constitutes the root cause, and to initiate reactions as to allow the network to continue its operations despite the belonging challenges.

5.3.6 Emergent Behaviors resulting from Software/Hardware Errors, Interoperability, Performance, Security related, and Automation Issues

Domain-FSSP: Resilience and recovery mechanisms, embedded inside the protocol modules and the services software, would have only limited capabilities of handling such faults. The reason for that is that the modules itself are causing the problems which constitute the emerging erroneous state.

Domain-FDH: In the long term, this type of faulty conditions should be resolved by improving the quality of the involved components. Such improvements can include bug fixing, supplemented by more extensive interoperability, penetration and performance testing. However, such procedures might be cost intensive and requiring a longer period of time until the new software releases are made available. In the period of these improvement activities, the FDH mechanisms (Fault-Mitigation running in parallel to Fault-Isolation and Fault-Removal) can provide a temporary solution which can enable the network to continue providing its services even in the case of such unwanted emergent behaviors. By supplying an enhanced Fault-Propagation Model and corresponding Fault-Masking/Removal policies considering the emergent faulty conditions in question, the human operator can enable the FDH framework in the nodes and the network as a whole to cope with such emergent situations, as long as the improved components are not available.

Chapter 6

Framework Specification

Having captured the key requirements for the emerging FDH, the current chapter proceeds with the design of the framework itself[1]. Thereby, the previously identified requirements are considered as a driving tool towards the specification of the distributed self-healing and belonging required components on node and network level.

At the beginning the distributed control is sketched, as to give the reader the rough idea of how FDH is meant to function. That includes outlining the interaction flows among the components dispersed along the network in question. That way, the reader is given a high level idea of the proposed approach without going into the "nuts and bolts" of the distributed FDH framework.

After presenting the network level view of the distributed FDH control loop, the focus is set on depicting the corresponding node level components, which are responsible for collaboratively realizing the process of self-healing. Having elaborated on each node level component, the dynamic aspects of the emerging framework are presented, i.e. the interaction flows among the various FDH components are described in detail. These interaction flows and process sequences achieve the FDH based self-healing as captured within the requirements analysis. In addition, the aspect of escalating faulty conditions, not resolvable by FDH means, to the network operations personnel is dealt with. This includes a set of criteria, which allow the FDH components across the network to determine on whether the distributed self-healing acts successfully, and if not, to escalate the situation and involve the network administrators.

6.1 Self-Healing Distributed Control Loop Design: High Level View

Within this section, the distributed self-healing control loop[2] is drafted without going into the details of the involved components. That way, a high level view of the proposed approach is presented before going into the detailed specification of each FDH component. Thus, the distributed FDH control loop is illustrated in Figure 6.1. First of all, it is presumed that each FDH node is equipped with an entity (Fault-Management Agent) that can potentially realize the processes of Fault-Isolation and Fault-Removal based on corresponding models, e.g. Fault-Propagation Models or Fault-Removal policy models. In addition, an FDH node should run a component (Survivabil-

[1] Initial considerations regarding the content of the current chapter were published in [TGV09], [TS14a], [CWM+13] and [WTVL13].

[2] The self-optimization features of FDH are left aside in this high level description as to convey the idea of the distributed self-healing control loop in a more simple way.

ity Agent) that would be responsible for Fault- Mitigation, i.e. immediate response to observed symptoms/indications for network problems. Another required assumption is that some of the nodes should have "monitoring sensors" deployed, i.e. monitoring components which are able to detect particular symptoms of faulty conditions. Given that one of these components has detected indications for an erroneous state, the symptoms are described in a specific format and initially pushed to the local Fault-Management and Survivability agents. Furthermore, they are conveyed to another local agent that takes care of disseminating the alarm/incident description within the network scope in question, e.g. an OSPF area or a LAN. That is, all the relevant nodes within the scope in question are aiming to converge with respect to their view on the set of symptoms detected across the network. Thereby, within each FDH node, the corresponding processes - inside the Fault-Management and Survivability agents - take place based on the converged view regarding the set of alarms/incidents detected across the network.

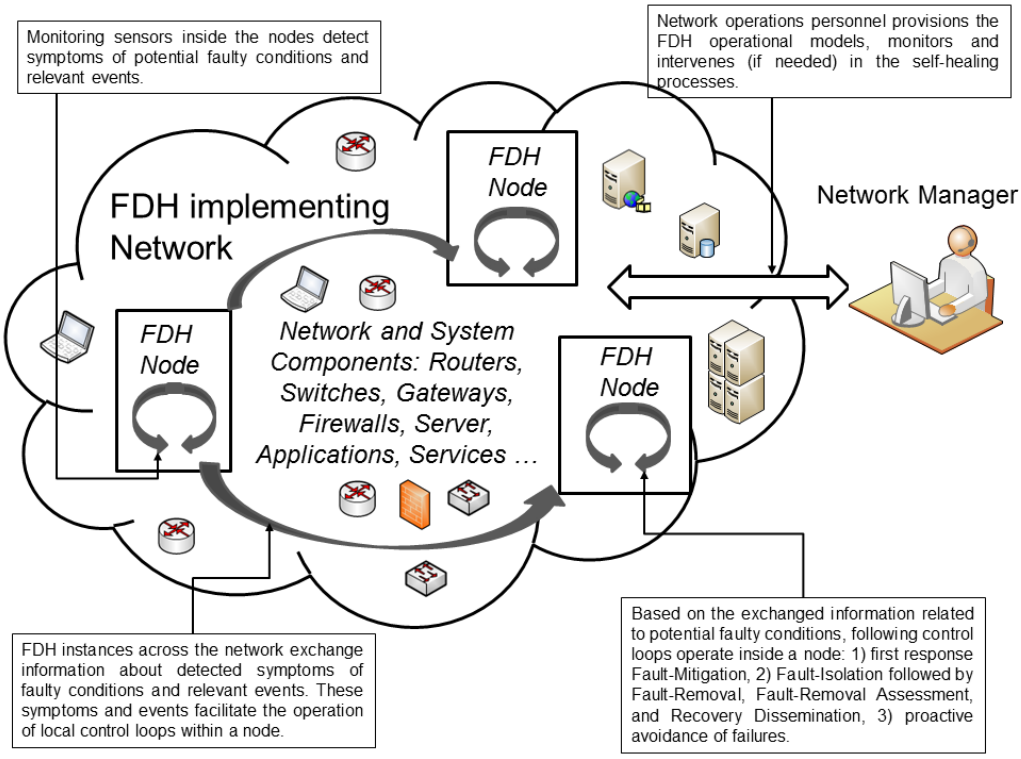

Figure 6.1: Overview of the Distributed Control Loop

The Survivability Agent in each node issues a node-specific Fault-Mitigation process, which is meant to locally address the immediate symptoms of the current fault(s) across the network. In parallel, the Fault-Management agents in each node periodically trigger a process of Fault-Isolation based on the set of symptoms which have been either detected locally, or have been received from the network.

The Fault-Isolation processes in all nodes run based on the same Fault-Propagation Model, a copy of which is stored locally within each node. This Fault-Propagation Model is provided by the network manager based on his anticipation for potential issues within the network in question - these anticipations can be based on experience, risk analysis, recent publications, or obtained by performing different tests. Given that all nodes have converged with respect to monitored symptoms (which might not always be the case), every Fault-Management Agent in each FDH node would

arrive at the same conclusion regarding the root cause(s) of the current erroneous state - due to the Fault-Isolation process based on the same Fault- Propagation Model deployed in each device. The results of this root cause analysis are then automatically forwarded to the Fault-Removal Functions (FRF) of each Fault-Management agent. This FRF module operates based on policies - also supplied by the network operations personnel - that are specific to every node in the network. That is, some of the nodes would issue an action as a response to the isolated fault and others would back off since their reaction is not required in the current situation. Finally, the effectiveness of the undertaken actions should be verified. Given that the actions have failed to achieve their goal, the situation has to be escalated and the network manager notified. In case that the faults have been successfully removed, a recovery notification should be sent across the network as to notify all FDH instances of the successful self-healing. The described steps hitherto indicate that the distributed control loop consists of local control loops, which are executed based on the incident information shared between nodes.

The above description clearly shows that the FDH framework is planned to execute a distributed control pattern, in which the the network nodes host the self-healing and self-optimization components. Hence, the implemented network control procedures do not rely on any centralized instance, which addresses **Req 2** from chapter 5. Furthermore, the proposed control loops are supposed to steer and manage every possible aspect of a network node with respect to incidents. Hence, it is presumed that they receive the required access to node operational aspects, which addresses **Req 2** from the requirements chapter. Based on the high level view presented above and the identified requirements, the next section proceeds with designing the components of FDH inside a node.

6.2 Components Specification on Node Level

Having presented the distributed self-healing control loop, the node level components of FDH are described within this section. Figure 6.2 illustrates the node architecture of the emerging framework. In general it consists of four different types of components:

1. components realizing the self-healing function
2. components which are mainly involved with the self-optimization features of FDH ("Self-Optimization Function" block in Figure 6.2)
3. various supporting components, which are in charge of incident dissemination/sharing, as well as of handling and storing different types of information and models that are vital for the control loop processes/phases, and finally
4. a set of components that serve as interfaces to the network operations personnel and are used, on one hand, for escalating (to the network operations personnel) challenges/situations not resolvable by FDH means, and on the other hand for the network administrators to provide the models, based on which FDH should operate

By devising and describing the above components, the current chapter addresses **Req 1** from section 5.1 which determines that FDH must define components and interfaces facilitating the tasks of Fault-Detection, Incident/Alarm-Dissemination, Fault-Isolation, Fault-Removal and Fault-Mitigation. These tasks and interfaces sketch the rough structure of the self-healing procedure that is specified in detail in the coming paragraphs. In this scope, the following subsections describe the nuts and bolts of the components illustrated on Figure 6.2.

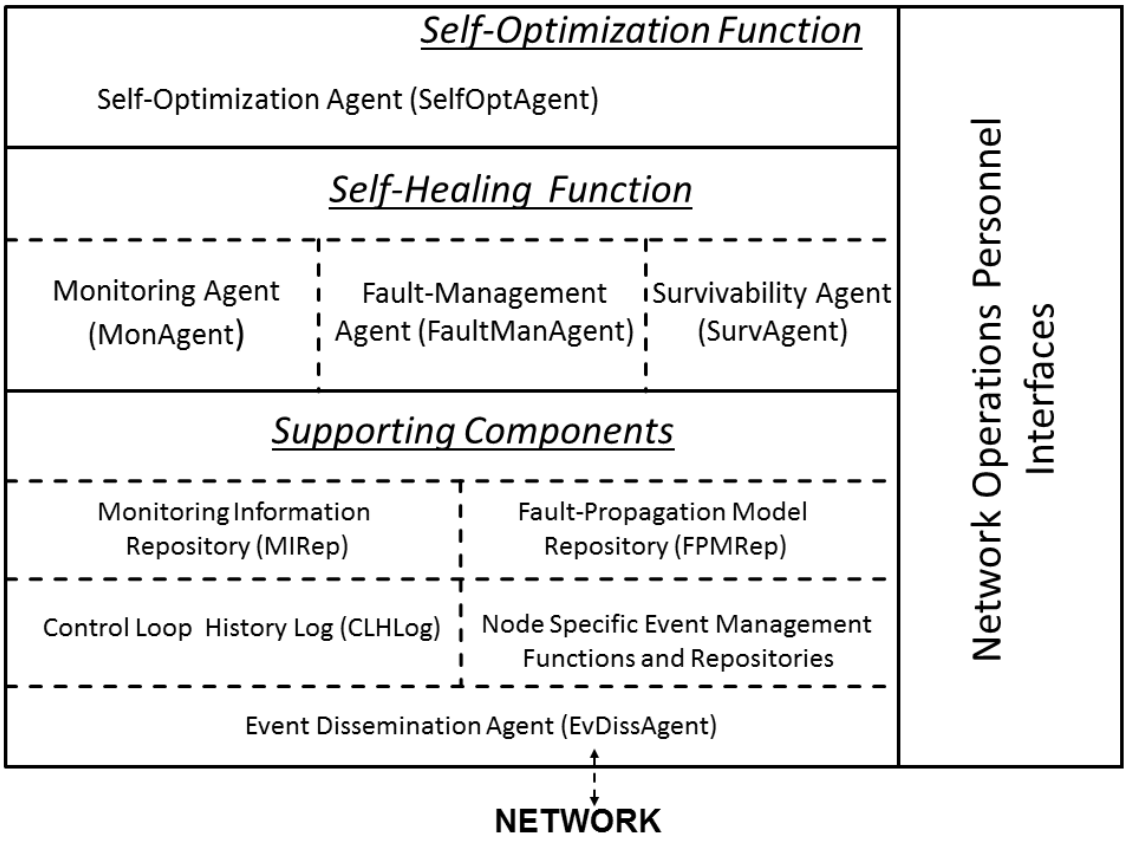

Figure 6.2: Overview of the Node Level Components of the Framework for Distributed Self-Healing

6.2.1 Self-Healing Function

The focus of the presentation is first set on the node components, which are solely responsible for the realization of the self-healing function. Thereby, the interactions with the Self-Optimization Agent, in charge of providing the self- optimization features of FDH, are pointed out. The elaboration starts with the monitoring functions of FDH and proceeds with the components that consume the outcome of the FDH monitoring framework and implement different reactions upon the detection of erroneous states. These components are given by the Survivability Agent, responsible for quick reactions and proactive avoidance of problems/challenges, as well as by the Fault-Management Agent that takes care of removing the root causes for the manifesting challenges in the long-term operation of the network /system.

6.2.1.1 Monitoring Agent

The entity responsible for coordinating the Fault-Detection efforts inside a network node is the Monitoring Agent (MonAgent) illustrated within the FDH entities on Figure 6.2 and depicted in detail in Figure 6.3. The Monitoring Agent is the main entity responsible for addressing the monitoring requirements in section 5.1.2, i.e. **Req 4-11**. Its role is to satisfy the monitoring requests and needs for tackling the erroneous states resolvable by means of FDH (i.e. **Req 4** and **Req 5**). That is, this entity starts monitoring jobs which observe the status of different opera-

tional key performance indicators (KPIs), and can recognize anomalous states which can be seen as symptoms of a potential fault within the network/system. Such KPIs might be for example the counters (e.g. dropped packets or frame collisions) of a Network Interface Card (NIC), different traffic metrics e.g. duplicate acknowledgments or retransmissions in TCP flows, as well as traditional QoS metrics as known within telecommunications, i.e. jitter, delay, out of order packets, packet loss (addressing **Req 11**). Thereby, the MonAgent would not only look for erroneous states but would also provide generic monitoring information which could be of use by any of the FDH components. This monitoring information is stored in a dedicated FDH repository within each FDH node (inline with monitoring requirement **Req 8**). The MonAgent orchestrates and regulates the behavior of different monitoring tools as to instrument passive and active distributed monitoring techniques (**Req 4**), in order to gain the required information. In addition, with respect to alarms/incidents/symptoms, there is also a direct communication channel to a set of lightweight FDH node-level repositories, which participate actively in the alarm/incident/symptom information sharing within a node and across the network scope in question. That is, once a symptom has been detected, it is described based on an information model and stored in the corresponding node-level FDH repository by the detecting component, which is orchestrated and instrumented by the MonAgent.

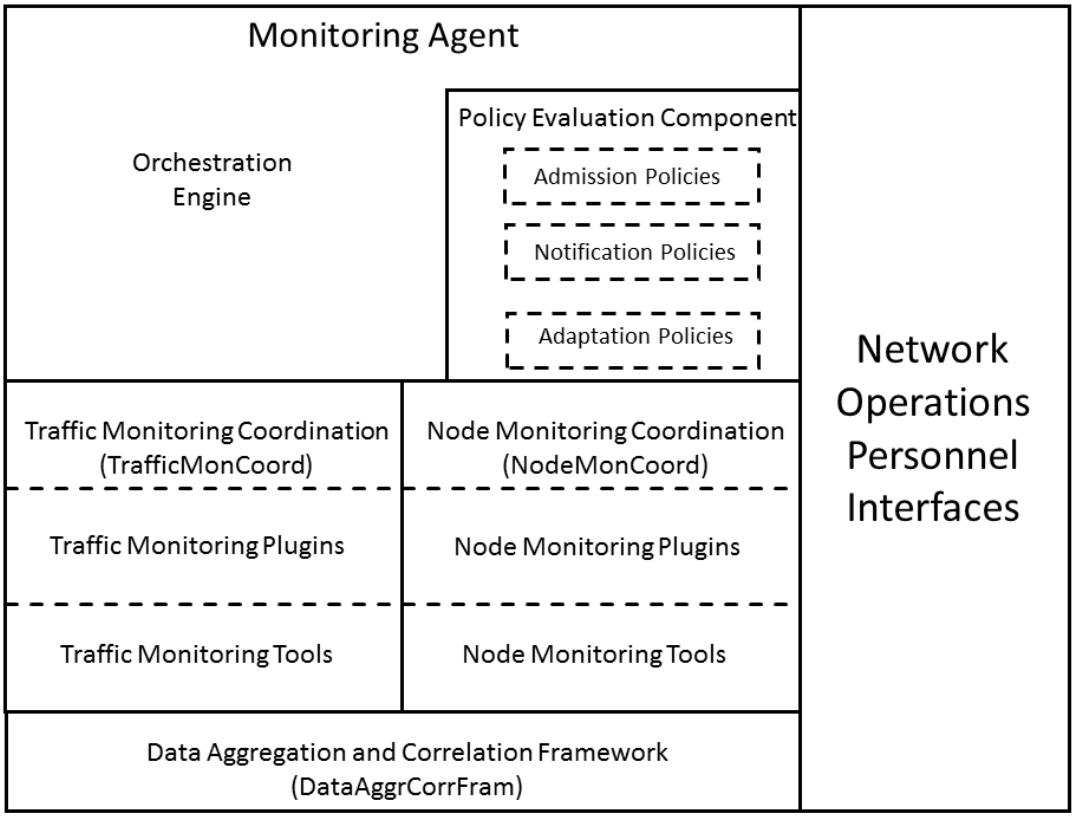

Figure 6.3: The FDH Monitoring Agent

Based on the above discussion, a number of components are specified which are needed to meet the identified requirements and the above elaborated aspects. The proposed monitoring architecture to serve the purposes of FDH is presented in Figure 6.3. It consists of four main components:

1. Orchestration Engine
2. Traffic Monitoring Coordination component
3. Node Monitoring Coordination component and a
4. Data Aggregation and Correlation Framework

In addition to these main components, a set of "Network Operations Personnel Interfaces" (on the right in Figure 6.3) are available. These interfaces serve the purpose of allowing the network administrators to configure various MonAgent aspects/parameters, as well as to obtain information regarding the operation of the FDH monitoring architecture in the network devices. Especially, these interfaces are utilized for supplying various policies which determine the operation of the FDH MonAgent within a node, i.e. addressing **Req 11** from section 5.1.2. Next, each of the four key components is described in turn with its associated sub modules, while the configurability over the "Network Operations Personnel Interfaces" is emphasized where appropriate.

Orchestration Engine: The *Orchestration Engine* is illustrated at the top in Figure 6.3 and constitutes the core of the *Monitoring Agent*, when it comes to processing monitoring requests and regulating monitoring services. This is the component that accepts monitoring requests (corresponding to **Req 6**) - either submitted by another FDH functional entity or by the network operations personnel - and decides on whether to accept them or not based on pre-configured *Admission Policies* - illustrated within the *Policy Evaluation Component* on the right side of Figure 6.3. Such admission policies are supplied by the network operator in advance over the corresponding interfaces part of the "Network Operations Personnel Interfaces", according to **Req 11**. An admission policy can, for example, reject the request in case memory is scarce at the given point in time. The evaluation of all types of policies within the Orchestration Engine, including the Admission Policies in question, is handled by the *Policy Evaluation Component*. Thereby, the action that would be issued as a result of the policy evaluation might need to be synchronized with the actions planned by the other FDH agents - the *Survivability Agent* and the *Fault-Management Agent*. Such coordination and action optimization between the FDH agents is handled by the *Self-Optimization Agent* described later in this chapter. In case a monitoring request has been allowed by both, the Admission Policies and the Self-Optimization Agent, the Orchestration Engine has to process the submitted request and trigger the corresponding *Traffic* or *Node Monitoring Coordination* component to setup the required monitoring service - inline with requirements **Req 6** and **Req 7**. In case of a problem while setting up the monitoring task, or if the request was not allowed, the Orchestration Engine has to respond to the requesting entity with the belonging error code or rejection notification.

In addition to the above aspects, the Orchestration Engine plays a vital role when processing the monitoring information obtained by the orchestrated monitoring tools. Once a measurement has been obtained, it is first forwarded to the *Data Aggregation and Correlation Framework*, which has the task to aggregate these measurements. Given that enough KPI values have been obtained and correlated, they are subsequently pushed back to the Orchestration Engine. The Orchestration Engine is then the module that is concerned with notifying the requester in question - FDH functional block or network administrator over the belonging "Network Operations Personnel Interfaces"- based on its embedded pre-configured *Notification Policies*, which are again evaluated by the *Policy Evaluation Component*. Thereby, the Notification Policies and resulting actions might also determine/drive the description of an alarm/incident based on the aggregated monitoring information and an underlying incident information model.

After the execution of the previously mentioned Notification Policies and resulting actions, the Orchestration Engine (correspondingly its Policy Evaluation Component) triggers and executes its pre-supplied *Adaptation* Policies that are in charge of adapting the behavior (e.g. sampling rate) of the monitoring task in question. Thereby, similarly as in the case of the Admission Policies, the action(s) resulting from the Adaptation Policies could potentially need synchronization (over the Self-Optimization Agent) with tentative actions planned by the other FDH agents. Within the above described process, both sets of policies - Notification and Adaptation Policies - are supplied by the network operations personnel over the belonging "Network Operations Personnel Interfaces" (i.e. **Req 11**).

The above descriptions clearly illustrate the realization of the adaptation control loop within the MonAgent. Thereby, the Orchestration Engine is the cornerstone for the realization of the control logic around the orchestrated monitoring services. It starts with receiving a monitoring request, and setting up a monitoring job, and continues with a number of actions after obtaining an aggregated KPI value. These actions include the notification to the requester based on the evaluation of some Notification policies, and finally the adaptation of the monitoring task based on some adaptation policies. The overall interplay of the MonAgent components within the control loop is later illustrated in the section describing the dynamic aspects of the MonAgent with the proposed FDH architecture.

Traffic Monitoring Coordination: The *Traffic Monitoring Coordination* (TMC) is the component responsible for setting up and running monitoring jobs related to network traffic passing through, or originating from the local node (i.e. **Req 7**). After a traffic monitoring request has been received and has passed through the admission control of the Orchestration Engine, the TMC module has to select the right plug-in for the submitted request, and to start it with the appropriate parameters. The selected traffic monitoring plug-in makes use of one or many traffic monitoring tools, which are combined as to deliver the required data.

Node Monitoring Coordination: The *Node Monitoring Coordination* acts similarly as the previously described TMC component with the difference that it is responsible for setting up monitoring jobs (i.e. **Req 7**). , which observe KPIs related to the state of the local node. As in the previous case, this component has the task of selecting the correct node monitoring plug-in and starting it with the right parameters upon a received monitoring request. As in the case of the TMC, the node monitoring plug-in makes use of one or many node monitoring tools, which are combined as to deliver the required data.

Data Aggregation and Correlation Framework (DAC): The Data Aggregation and Correlation Framework (DAC) plays and important role in the course of data aggregation (inline with monitoring requirement **Req 9**) and alarm/incident/monitoring information suppression, which are among the vital challenges of traditional network management. It is presumed that each monitoring plug-in is associated with a specific instance of DAC. These DAC instances are responsible for aggregating the data for a certain monitoring plug-in. There can also be different aggregation mechanisms inside the instances, e.g. one can be using a smoothing average filter whilst another one is looking for the median of a number of monitored samples. Given that data aggregation has been accomplished, the summarized value is pushed to and processed by the Orchestration Engine.

6.2.1.2 Survivability Agent

To support generic Resilience and Survivability functions within an FDH implementing node, the *Survivability Agent* (SurvAgent) is proposed as a core part of the presented FDH node architecture. The components encompassing the SurvAgent are shown in Figure 6.4. The two main components are given by the *Fault-Mitigation Functions* (FMF) and the *Failure-Prediction Functions* (FPF).

The FMF is the SurvAgent module that takes care of first immediate reaction to the symptoms of an erroneous state, while at the same time the Fault-Management Agent is concerned with resolving the root cause(s) of the problem in the long-term operation of the network.

In parallel, the FPF consume various types of monitoring information in an effort to identify potential failures in near future. This would result in a warning alarm upon which the FMF should take an action to mitigate the potential faulty condition. In the course of their operation, the FMF and FPF depend on information from the MonAgent and the monitoring plug-ins operating under its orchestration. This information is stored in the local lightweight FDH repositories and forwarded to the SurvAgent. Similarly as in the case of the MonAgent, both components are configured and monitored over a set of "Network Operations Personnel Interfaces".

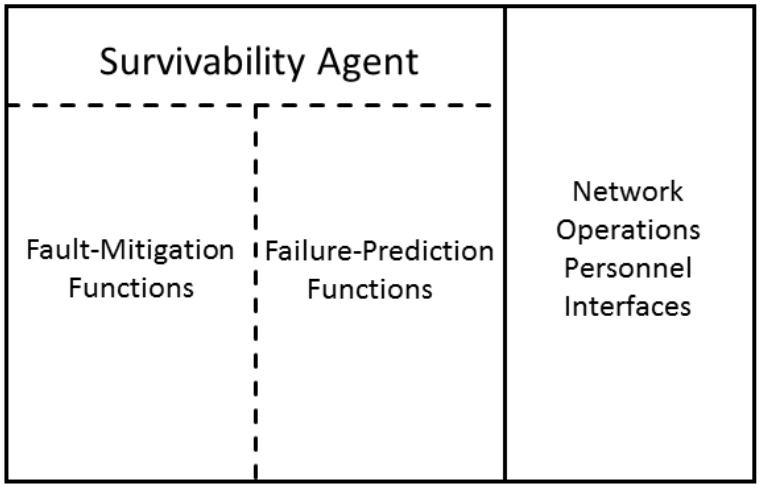

Figure 6.4: The Survivability Agent

Fault-Mitigation Functions: Having information from the FDH node repositories, the SurvAgent triggers diverse reactive fault-masking mechanisms depending on the situation in the network. As previously mentioned, this process is realized by the Fault-Mitigation Functions (FMF) module towards realizing fast immediate reactive resilience within the FDH nodes. The role of the SurvAgent is to trigger specific Fault-Mitigation actions that would be best applicable as an immediate reaction to the current incident(s), alarm(s) and symptom(s) in general . However specific fault tolerant algorithms/procedures/behaviors are not implemented within the SurvAgent and its FMF module, but their functionality is specific to the underlying lower level functional entities (e.g. protocols, network layer restoration schemes, etc.).

After having been informed about an incident, the SurvAgent synchronizes its tentative action(s) with the other agents of the FDH node architecture, i.e. the Monitoring Agent and the Fault-Management Agent. This synchronization is conducted over the Self-Optimization Agent presented in the next section. Based on the synchronization result, the SurvAgent discards or per-

forms the Fault-Mitigation action(s) on the corresponding managed resources, whilst taking into consideration their embedded fault-tolerant behaviors. Thus, the SurvAgent acts in a fault-tolerant manner and orchestrates immediate resilient mechanisms involving the selected node entities to ensure acceptable QoS and sustain network functionality. In order to select appropriate solutions towards a proper first response resilient behavior of a node, the SurvAgent must be aware of the resilient/fault-tolerant mechanisms implemented intrinsically inside the entities, which are under its control, e.g. protocol modules. In this line of thought, before imposing its own decisions, the SurvAgent is required to give the low level entities a chance to recover based on their own intrinsic resilient mechanisms. However, first response responsibilities belong to the SurvAgent and thus it should have the right and means to intercept the intrinsic actions, undertaken by the node functional entities if required.

Failure-Prediction Functions: While the FMF module of the SurvAgent takes reactive actions upon detected incidents, the Failure-Prediction Functions (FPF) module tries to anticipate the occurrence of certain incidents in the near future. As a result, the SurvAgent is able to proactively steer the node entities such that the node and/or network can prepare for the problems that are likely to manifest in the future. As a consequence, it is possible to avoid future fault activations. The FPF module enables the Fault-Management and Survivability agents to shift from basic, reactive only schemes, to new proactive ones where prediction allows the framework to anticipate failures as well as to dynamically improve its operation, e.g. the Fault-Propagation Model within the FDH nodes may be adapted based on the anticipated faulty conditions.

6.2.1.3 Fault-Management Agent

The Fault-Management Agent (FaultManAgent) agent within a node is responsible for the automation of Fault-Detection, Fault -Isolation and Fault-Removal with respect to faulty conditions which affect the local device. The internal architecture of the FaultManAgent is depicted in Figure 6.5. In general, the automation of the above processes (Fault-Detection, Fault-Isolation, and Fault-Removal) is assigned to the Fault-Isolation Function (FIF) - being the component gluing Fault-Detection and Fault-Isolation - and the Fault-Removal Functions (FRF) for removing faults and their effects locally (after a successful Fault-Isolation). In addition, as these Fault-Management processes get automated and glued together, the (intermediate) outcomes of Fault-Isolation and Fault-Removal need to be verified. The task of checking the correctness of the Fault-Isolation results is realized by the Fault-Isolation Assessment Functions (FIAF), and the task of verifying the effectiveness of the Fault-Removal actions is realized by the Fault-Removal Assessment Functions (FRAF). Next, all these components are described in turn.

Fault-Isolation Functions: The *Fault-Isolation Functions* (FIF) module is the component responsible for isolating and diagnosing faults in the process chain of the automated Fault-Management implemented by the FaultManAgent. The FIF functions are realized by a library which is invoked by the FaultManAgent after some incidents (or alarms) have been reported and their analysis is correspondingly required. A submitted incident (or alarm/symptom/event) [3] description can be either referring to locally detected incidents which are forwarded to the FaultManAgent agent by the corresponding repository or to incidents which were detected elsewhere in the network and were reported locally using the service of the Event Dissemination Engine (see section 6.2.3.2). The FIF functions gather such incident descriptions and trigger the Fault-Isolation process according

[3]The term *incident* is quite often used in the broader sense encompassing all types of possible artifacts such as events, symptoms, alarms and incidents relating to a faulty condition

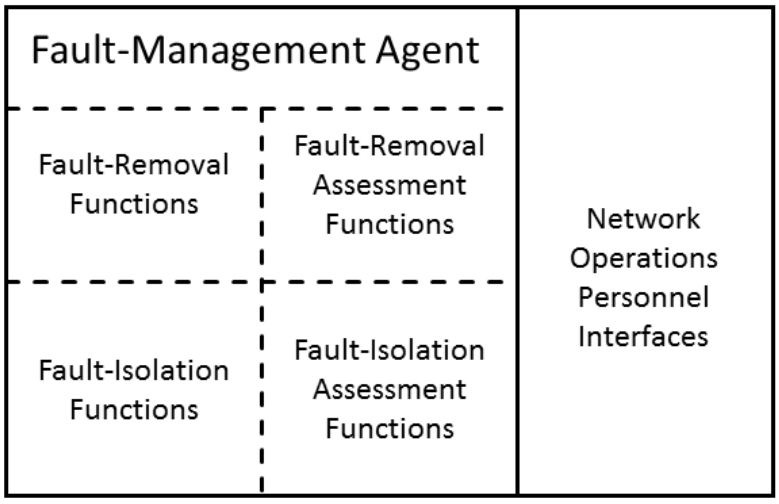

Figure 6.5: The FDH Fault-Management Agent

to some pre-defined conditions:

1. Fault-Isolation is triggered every time a particular pre-defined number of incidents (or alarms/symptoms/events) have been reported

2. Fault-Isolation may also get immediately initiated after a single incident event has been reported given that the event has a high (according to a pre-defined value) severity level and

3. if only few (less than the pre-defined number of) incidents/alarms/events/symptoms are reported and their severity is not causing the FIF functions to immediately trigger Fault-Isolation, then Fault-Diagnosis is triggered after a pre-defined time slice, which can be realized by the expiration of a timer.

The aforementioned number of incidents, the severity level of the incidents immediately triggering Fault-Isolation, and the duration length of the time slice after which the FIF functions initiate Fault-Isolation, are parameters, which need to be supplied by the human experts tweaking the FDH operation. Once Fault-Diagnosis is triggered, selected algorithms start reasoning based on the knowledge[4] regarding the relations between faults/errors/failures/alarms in the network, captured in a dedicated node repository denoted as Fault-Propagation Model Repository (FPMR) and described in the next sections.

Fault-Isolation Assessment Functions: Given the uncertainties which accompany the task of Fault-Isolation, it is imperative to ensure that the results of the Fault-Isolation process are indeed correct - as required in **Req 18** in section 5.1.4 - before issuing a Fault- Removal action. These uncertainties arise due to the stochastic nature and complexity of the network and deployed protocols/services in a distributed environment. Hence, the Fault-Isolation process has to intrinsically rely on probability based models (e.g. Bayesian Networks) for its reasoning. Thus, its results might also be wrong and the resulting Fault-Removal actions could potentially even worsen the erroneous state of the network. The Fault-Isolation Assessment Functions (FIAF) is the component responsible

[4]This type of knowledge is denoted as Fault-Propagation Model and fulfills **Req 17** from section 5.1.4

for avoiding such situations. In the course of its operation, the FIAF can for example deploy different probing mechanisms, check the running configuration of some entities, examine the queue sizes or the state of the node, in order to test and verify the results of the Fault-Isolation task.

Fault-Removal Functions: Given the verified results of the Fault-Isolation process, the *Fault-Removal Functions* (FRF) of the FaultManAgent provide the means by which a specific fault can be removed. The FRF of a device become activated when the isolated fault is related either to the node itself or to its neighborhood, and the reaction of the node can remediate the erroneous state of the network. The FRF module implements functionalities that influence the behavior of the pool of functional entities in the network, and therefore is a component that is critical with respect to the stability and overall effectiveness of the FDH control loops (SurvAgent, FaultManAgent, and MonAgent control loops). In order to ensure stability with regard to avoiding contradicting actions issued by parallel instances of the FDH control loops, the FRF refers to the local Self-Optimization Agent. The Self-Optimization Engine is responsible for synchronizing such tentative actions, and for selecting an optimal subset (from the tentative actions) based on the goal of improving the performance of the network/system. The Self-Optimization Agent component is elaborated in more detail in the coming sections. Given that an action has been allowed by the Self-Optimization Agent, the FRF triggers the execution of the corresponding scripts or management tools, for instance by using their belonging CLI (Command Line Interface). All the information regarding the execution of an action can be captured in a policy like information model (derived) which is supplied by the human experts tweaking the FDH processes of the network.

Fault-Removal Assessment Functions: Finally, the Fault-Removal Assessment Functions (FRAF) have the task to inspect whether the Fault-Removal process within the FaultManAgent, following the process of Fault-Isolation, was successful or not (thereby explicitly fulfilling the belonging **Req 21** from section 5.1.4). Hence, the FRAF realize functions like probing and testing, in order to verify that a functionality or service has been successfully restored and is available again. Based on the result from this assessment, a local FRAF escalates the problem to the network operation personnel in case Fault-Removal has failed. Alternatively, the FRAF marks as cleared/recovered all the incidents and alarms related to the problem in question, in case Fault-Removal was successful. At the end, the control loop's execution, with its status obtained by the FRAF, is logged to the local Control Loop History Log described in the coming sections.

6.2.2 Self-Optimization Function

The other main pillar of FDH, besides the distributed self-healing, is given by the self-optimization features the framework implements. These features are mainly realized by the Self-Optimization Agent described in the following subsection.

6.2.2.1 Self-Optimization Agent

The Self-Optimization Agent (SelfOptAgent) is the component responsible for ensuring the stability and overall efficiency of the various FDH control loops executing in parallel. These control loops are implemented within the MonAgent, the FaultManAgent, and the SurvAgent within an FDH node. Architecturally, the SelfOptAgent consists of a module that implements the action synchronization and optimization function which is configured for each FDH instance over a set of *Network Operations Personnel Interfaces*.

The task of the SelfOptAgent is to "gather" tentative actions and to synchronize them towards a common goal based on a pre-defined utility function and optimization problem. Similarly as in the case of the FIF part of the FaultManAgent, the tentative actions are gathered over a pre-defined period of time after which the synchronization is initiated. A synchronization of tentative actions can be also triggered after a pre-defined number of synchronization requests have been submitted, or in the case when a synchronization request is marked as urgent. The SelfOptAgent allows only those tentative actions to be executed that do not interfere with each other and contribute to the optimization of the corresponding utility function, i.e. maximization of the underlying network fitness model. Hence, the SelfOptAgent is naturally defined as the component fulfilling **Req 20** from section 5.1.4 within the requirements chapter. More details on the algorithms for self-optimization and the applied structures can be found later in section 8.

6.2.3 Supporting Components

The previously described Self-Healing and Self-Optimization functions require various supporting components, in order to operate as required and specified. These components allow the FDH instances to store, exchange, and manage information which is required for the different FDH control loops within a node and the network as a whole. The information in question includes alarms and incidents that were submitted to FDH by different monitoring entities. Furthermore, the supporting components - described in the following paragraphs - store models which are needed in the scope of Fault-Isolation, i.e. node-specific Fault-Propagation Models. In addition, the history of control loop executions is also correspondingly tracked as to allow the network administrators to follow and understand the operation of the FDH instances across the network, as well as to enable the awareness of different FDH functions (e.g. FRF) with respect to the history of FDH control loop actions. The components described below are vital for the realization of the FDH based distributed self- healing since they provide the platform, on the top of which the FaultManAgent, the SurvAgent, and the MonAgent realize their control loops.

6.2.3.1 Node-Specific Event Management Functions and Repositories

The overall set of *Node-Specific Event Management Functions and Repositories* is depicted in Figure 6.6. The various modules and their role in the distributed control loops are described in the following paragraphs. Thereby, it should be noted that all modules implement a set of interfaces (on the right side in Figure 6.6), which are accessed by the network operations personnel, in order to monitor or configure a particular part of the *Node-Specific Management Functions and Repositories*.

Node Repositories: The alarm/incident repositories (in the middle of Figure 6.6) inside the device architecture are the consolidating point for faults/errors/failures/alarms (and symptoms for faulty conditions in general) coming from the node and from the network scope in question (e.g. LAN or OSPF area). The information is stored in a structured way in different repositories, so that a classification of incidents from the point of view of the local node is achieved. In that regard, the FDH design borrows a classification from previous work [Cha09] [CTS08] related to Failure Detection in clean slate networks, and adopts the following repositories inside an FDH instance:

1. Local and External Alarms Repository
2. Local Incidents Repository

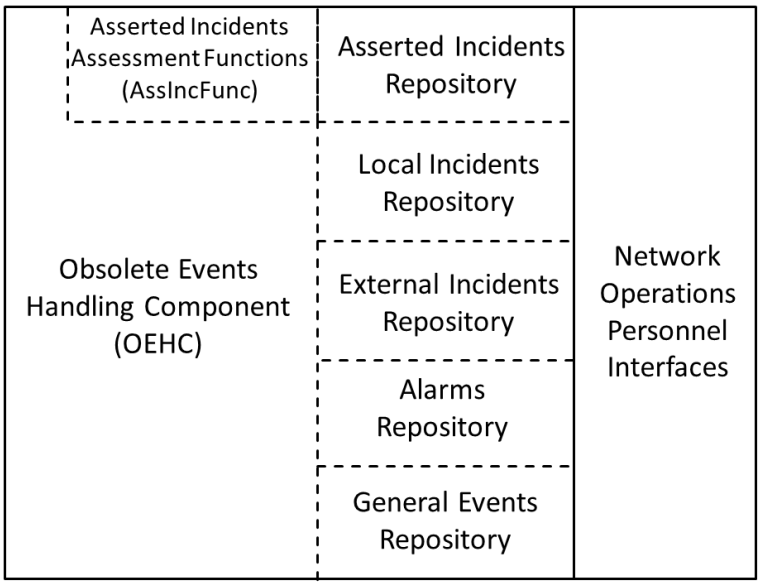

Figure 6.6: FDH Node-Specific Event Management Functions and Repositories

3. External Incidents Repository - dedicated to storing incident information related to other network nodes

4. Asserted Incidents Repository - responsible for storing incident information coming from other network nodes and related to the local node

Furthermore, a repository for general events - i.e. *General Events Repository* is required in order to allow the FDH taking notice and considering situations such as scheduled maintenance or any other types of events, which are not really an incident but should be taken into account within the control loops' executions.

Regarding the involvement of the FDH node repositories in the alarm/incident information exchange: the repositories are responsible for forwarding submitted alarm/incident descriptions to the local FaultManAgent and the SurvAgent. In that sense, the repositories fulfill requirement **Req 12** from section 5.1.3 that specifies the need for collaborative alarm/information sharing across a single node and the network as a whole. Indeed, once a symptom has been detected by the corresponding monitoring entity (orchestrated by the MonAgent) and submitted to the belonging repository, the alarm/incident description is automatically forwarded to the local Fault-Management and Survivability agents, as well as to the local Event Dissemination Agent, such that it gets disseminated across the network scope in question (e.g. LAN or OSPF area). In case a fault has been removed and a recovery notification has been issued by the local FaultManAgent or received over the Event Dissemination Agent - in case the fault was removed by a FaultManAgent on a remote FDH node - the repositories should schedule the related incidents for deletion by the *Obsolete Event Handling Component*, which is described below.

Asserted Incidents Assessment Functions: The incident information being reported to the FDH node repositories may trigger local probing mechanisms, in order to verify whether a particular functionality or service offered by the local node is still available and functioning correctly. This functionality is realized by the *Asserted Incidents Assessment Functions* (AssIncFunc) which is a separate agent that triggers different probing mechanisms based on a reaction model supplied

by the network operations personnel. In case the AssIncFunc have found out that a particular functionality is actually available, contrary to the suggestion of a particular asserted incident, a recovery notification for this particular incident would be disseminated using the Event Dissemination Agent. Subsequently, the incident would be marked as recovered in the FDH repositories across the network and would be scheduled for deletion by the *Obsolete Event Handling Component*, which is described next.

Obsolete Event Handling Component: The *Obsolete Event Handling Component* (OEHC) is an integral part of the alarm/incident repositories, and as the name suggests, it is responsible for removing and dumping alarm and incident information, which relates to already resolved erroneous states. In the course of experimenting with the FDH prototype in the testbed presented in section 11, a number of situations were encountered where, due to delays in the convergence of the dissemination mechanisms (within the Event Dissemination Agent), information about a particular incident event arrived at a node after the incident has been recovered and locally deleted from the repositories by the Fault-Removal and Fault- Removal Assessment mechanisms. This disorder of notifications has led to repeatedly triggering Fault-Isolation with a wrong set of detected incidents. In order to resolve this, the incident/alarm events are not immediately removed when a recovery notification is issued by the FRF and the FRAF mechanisms, but their status is set to recovered/cleared. Therefore, if a particular incident detection notification arrives after the corresponding incident recovery notification, all the means (in terms of information) are available to resolve this ambiguous situation. However, at some point the information about the recovered incident has to be deleted from the node repositories and saved to the log history of the framework. This is where the *OEHC* is required. *The OEHC checks periodically for alarms and incidents having the cleared/ recovered status inside the repositories, and dumps this information to a log thereby deleting it and releasing the allocated memory within the repository processes.*[5]

6.2.3.2 Event Dissemination Agent

The *Event Dissemination Agent* (EvDissAgent) is the entity that plays the role of a gateway to the network inside an FDH implementing device, thereby addressing **Req 12** from section 5.1.3. It is responsible for disseminating and receiving alarm/incident/events information (e.g. symptoms for faulty conditions) and recovery (i.e. Fault-Removal) notifications to and from the network. Thus, once a symptom of a faulty condition has been observed, described and submitted to the incident repositories on the node hosting the corresponding Fault-Detection component, the repository in question forwards the incident description to the FaultManAgent, the SurvAgent and to the local EvDissAgent. In that context, alarms are forwarded and processed exactly the same way. The EvDissAgent in turn disseminates the alarm/incident description to involved nodes across the network scope where the corresponding FDH control loops are to be triggered based on the alarm/incident information. With respect to receiving alarm/incident information from the network, the local FDH EvDissAgent has the responsibility of storing it in the appropriate local node alarm/incident repository and conveying it to the local Fault-Management and Survivability agents, such that this incident description can be considered for the following FDH control loop executions. Finally, after a successful completion and assessment of a Fault-Removal process, a recovery notification is sent (by the corresponding FaultManAgent) across the network scope (over the services of the EvDissAgents) such that the relating symptom descriptions can be deleted across the repositories of the involved nodes.

[5] The term process relates to an operating system process.

6.2.3.3 Monitoring Information Repository

The *Monitoring Information Repository* (MIRep) is used to store a history of the obtained monitoring information within a device[6]. It keeps track of important KPI values measured by the monitoring plug-ins of the MonAgent, such that these values are available for different parties, e.g. for the network operations personnel, for the FRF/FIF/FIAF/FRAF modules of the FaultManAgent, for the FMF/FPF modules of the SurvAgent, or for the diverse policies of the Orchestration Engine within the MonAgent.

6.2.3.4 Fault-Propagation Model Repository

The *Fault-Propagation Model Repository* (FPMR) is the FDH node component responsible for storing a local node instance of the Fault-Propagation Model (FPM) for the network scope in question (e.g. LAN or an OSPF area). The FPM stores the causality relations between different symptoms (failures and alarms), intermediate unobservable events in a fault- propagation i.e. errors, and the corresponding root causes (faults). These relations are stored in a way that allows fast real-time reasoning and root cause analysis. At this point it should be mentioned that the Fault-Propagation Model and the FPMR facilitate addressing **Req 17** from section 5.1.4. Indeed, the FIF functions of the FaultManAgent are accessing the FPMR and the stored FPM instance every time they need to perform Fault-Isolation. The contents of the FPMR are supplied by human experts that have in-depth knowledge of potential cases of fault-propagation in the network, depending on the deployed hardware and software. The algorithms, which are used by FDH to query the FPM, are detailed in section 7.3.

6.2.3.5 Control Loop History Log

Naturally, all control loop activations, including the intermediate results produced within the different involved components, should be recorded. These records are stored in the *Control Loop History Log* (CHL) within each FDH node and can serve different purposes. Within a possible utilization, the CHL records can be used by the network operations personnel to understand and improve the operation of FDH within the network. For example, the Fault-Propagation Model may be improved as to deliver better Fault-Isolation results, or the different policy models specifying the actions in different stages of the control loop (Fault-Removal, Fault-Isolation Assessment etc.) can be further refined based on the recorded logs. The control loop history should also be easily accessible over an API as to give the FaultManAgent-FRF functions (and various policy based modules in general) the capability to be aware of the type of actions which were already executed by the FDH - inline with the need identified in **Req 19** from the requirements chapter. The experience, drawn from experiments with the FDH prototype, has shown that this can be especially helpful in situations where an entity (e.g. a NIC or its driver) is not really stable and multiple reconfigurations in a row make it unusable, thus requiring the hardware/software component to be restarted or completely reinitialized.

[6] In this scope, the *MIRep* is responsible for fulfilling **Req 8** from the monitoring requirements.

6.2.4 Network Operations Personnel Interfaces

The Network Operations Personnel Interfaces stand for a set of components that enable human experts to steer and monitor the operation of the FDH framework across the network, and that way to effectively handle faults/errors/failures/alarms. Basically, each FDH component within the FDH node architecture on Figure 6.2 possesses various interfaces that allow the network administrator to configure it and monitor its operations. Figure 6.7 provides a unified view on the different modules enabling the interactions between the network administrators and the FDH components across the network. Next, each of these modules is described in turn.

Bootstrap Manager: A *Boot Manager* resides in each FDH node and takes care of the correct bootstrapping of the self-healing components in a network device. The Boot Manager is responsible for starting all the FDH modules on Figure 6.2 in the right order. Thereby, it monitors the status of the FDH components during the bootstrapping of a node and reports any occurring issues within the process to the network operations personnel.

Provisioning Tools: Different models are required for the operation of the distributed self-healing framework across the network. This includes the network-wide Fault-Propagation Model as well as various node-specific models, such as the Fault-Removal policies, the Fault-Isolation Assessment policies, the Fault-Mitigation reactions policies etc. All these models must be designed by the administrators for the network as a whole as well as for the individual FDH nodes. In order to support the design of these models, a set of provisioning tools is required, which would aid the network operations personnel to efficiently create the required FDH models. Subsequently, these models are stored in a centralized *Configuration Server*, from where they can be downloaded to the individual nodes during the bootstrapping phase of an FDH node.

Configuration Server: As previously mentioned, the centralized *Configuration Server* is the component that stores the FDH operational models for the network and the single nodes, once these models have been designed using the above elaborated *Provisioning Tools*. During the bootstrapping phase of each device, the single FDH components build up a connection to the centralized Configuration Server and download the belonging models required for the realization of FDH based self-healing. Thereby, the downloaded models are locally stored in the corresponding repositories.

Monitoring Interfaces to the Network Nodes: The FDH network nodes provide monitoring interfaces, over which the operation of the FDH components inside a device can be observed. These monitoring interfaces can be differently realized, e.g. by using SNMP or CLI. Moreover, the FDH control loops log the different steps of their execution to the Control Loop History Log on each FDH node. These logs are also accessible for the purpose of monitoring the self-healing control loops across the network. The monitoring interfaces in question are also used for escalating faulty conditions, which are not resolvable by FDH means, to the network operations personnel. The aspects of faulty condition escalations were ellaborated in the previous sections, e.g. in the part relating to the FRAF functions of the FaultManAgent.

Network Monitoring System: The various aspects of the FDH operations need to be presented to the network operations personnel in a comprehensive manner. This should be done by integrating the above described monitoring interfaces into existing *Network Monitoring Systems* (e.g. Nagios [Bar08]). That way the network administrators will have the possibility to oversee the self-healing activities across the network and get involved if required, e.g. the human experts can improve the FDH operations by adapting the operational models. The Network Monitoring System is also the component which presents, to the network administrators, information related to faulty conditions

that are not resolvable by FDH means and are being escalated because of that.

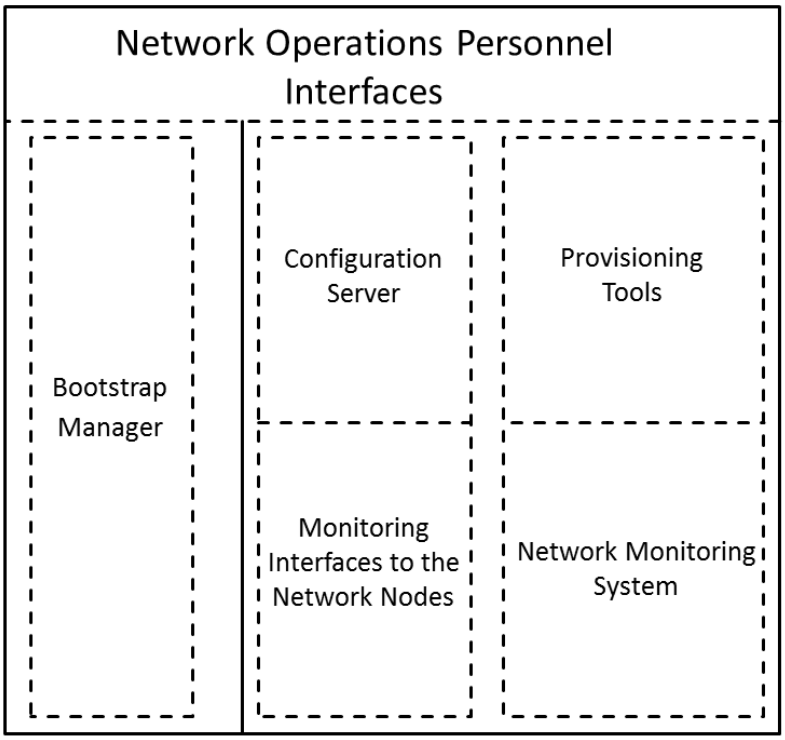

Figure 6.7: FDH Network Operations Personnel Interfaces

6.3 Dynamic Aspects of the Framework

Based on the architectural descriptions provided above, the current section proceeds with outlining the dynamic aspects of the FDH framework and specifying the distributed control loop in detail. In the course of this, the various interaction flows among the FDH components are described using the concept of UML sequence diagrams. The first set of interactions, which need to be elucidated, are those pertaining to the the bootstrapping phase of an FDH node. These interactions are of relevance to the network operations personnel activities relating to setting up a functioning FDH based self-healing infrastructure in a network. However, given that the bootstrap sequences are not at the heart of the self-healing and self-optimization control loops, they are described in detail in the appendix of this thesis (i.e. in appendix section B.1) and not in the main thesis part regarding the architectural design. Hence, the current section starts directly with describing the dynamic aspects relating to the tasks of Monitoring and Fault-Detection, which corresponds to the first step of the FDH control loops. Subsequently, a set of sequence diagrams are presented that depict the process of how an incident/alarm description is managed and made available for the node level self-healing machinery. In that context, the sequence diagrams also elaborate on how an incident/alarm description plays a role within the distributed self-healing on network level. Finally, the processes implemented within the Fault-Management Agent and the Survivability Agent are described, which aim at mitigating the immediate impact of erroneous states, whilst at the same time removing the underlying root cause(s) in the long-term operation of the network.

In addition to the dynamic aspects presented in this section, appendix F introduces further dy-

namic aspects which are not the main focus of this thesis. These aspects are mainly related to the collaboration of FDH instances across multiple domains, e.g. among FDH instances embedded in end-systems, access, edge and core network components.

6.3.1 Monitoring and Fault-Detection

The sequence diagram on Figure 6.8 illustrates the processing of monitoring information resulting from a previously instrumented monitoring job. The corresponding monitoring plug-in combines the outputs of different monitoring tools as to extract some relevant information (*orchestrateMonitoringActivities*). At this point, it is possible that the plug-in in question detects a fault and directly pushes it to the local node repositories of the FDH framework.

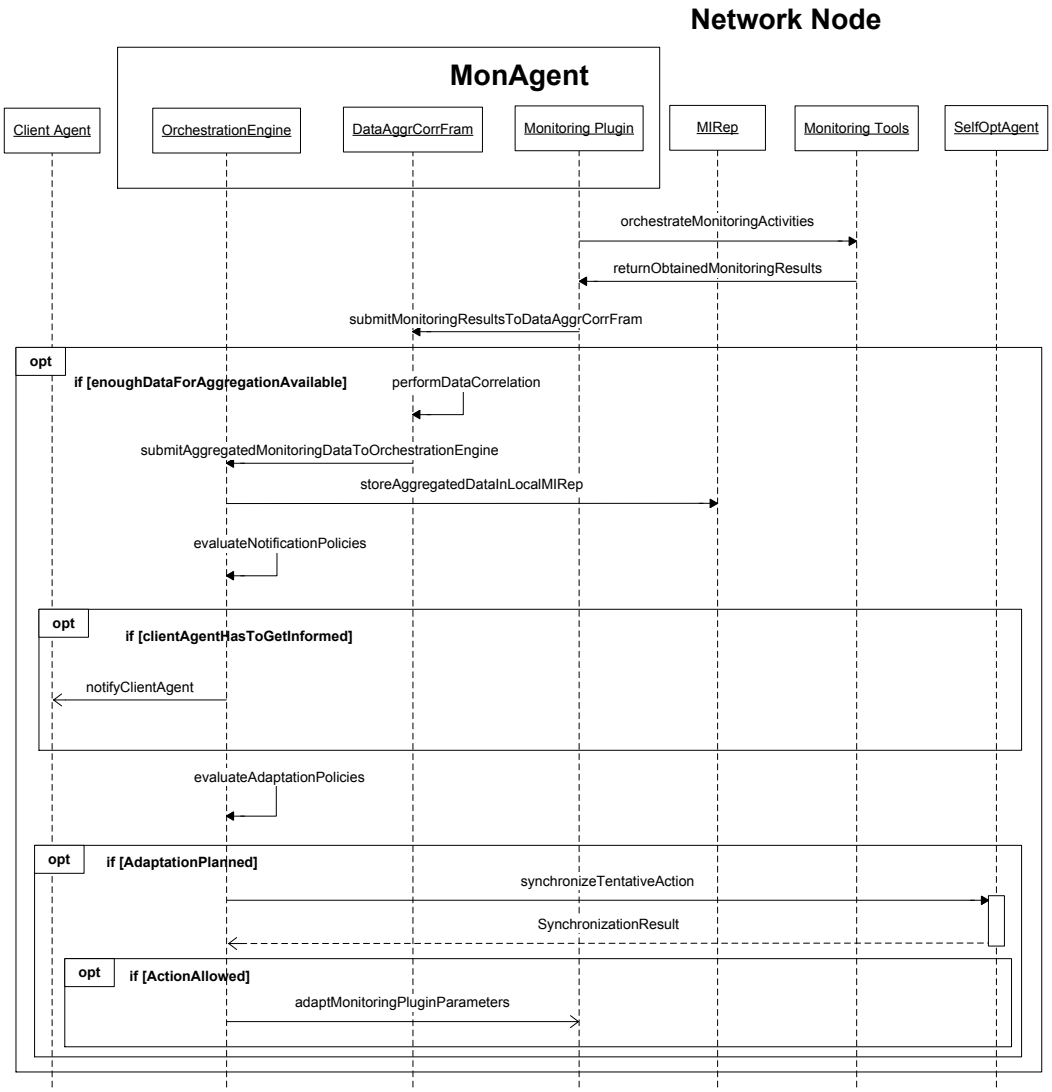

Figure 6.8: Processing Monitoring Information

Besides the latter possibility to submit an incident/alarm directly from a monitoring plug-in to the local FDH repositories, it is also possible to pursue the general path depicted on Figure 6.8, where

the results obtained by the monitoring plug-in are forwarded (*submitMonitoringResultsToDataAggrCorrFram*) to the *DataAggrComFram (Data Aggregation and Correlation Framework)* that in turn aggregates (*performDataCorrelation*) several such measurements before reporting (*submitAggregatedMonitoringDataToOrchestrationEngine*) the aggregated value to the *Orchestration Engine* of the MonAgent. Subsequently, the Orchestration Engine stores (*storeAggregatedDataInLocalMIRep*) the aggregated data into the local FDH repository for monitoring data - i.e. *MIRep* - and triggers the evaluation of the notification policies (*evaluateNotificationPolicies*). These preconfigured polices determine whether the aggregated value should be sent to the requester and potentially trigger its control loop. The notification policies would also be the place where an alarm might be generated, or an incident description might be created, in order to push it to the local FDH alarm/incident/event repositories thereby completing the process of Fault-Detection. To summarize: On one hand, the notification policies play an *"alarm suppression"* type of a role, since they push only vital monitoring information to the agents of the FDH framework. On the other hand, the notification policies constitute the hook - besides the possibility to implement Fault-Detection or Alarm Generation directly in a monitoring plug-in - for realizing Fault-Detection and Alarm Generation based on aggregated monitoring KPIs provided by the plug-in components.

After the monitoring information has been recorded to the local monitoring repository and the *Notification Policies* have been evaluated and executed, the pre-configured *Adaptation Policies* are triggered (*evaluateAdaptationPolicies*), in order to modify the behavior of a monitoring job if required. The *Adaptation Policies* constitute the hook for realizing regulative mechanisms with respect to the running FDH monitoring jobs. Indeed, this is the place where aspects such as sampling rate, aggregation strategy and resource allocation around a monitoring service/job/task, can be all adapted. Thereby, the adaptation of the monitoring job might need to be synchronized (*synchronizeTentativAction*) over the *SelfOptAgent*, since it constitutes an action that would potentially require additional resources and might contradict with other tentative actions originating from the FDH agents within a node. Given that the action has been allowed by the SelfOptAgent, the adaptation of the belonging monitoring parameters (e.g. sampling rate) is conducted (*adaptMonitoringPluginParameters*). In Appendix E, a specific example of how such an adaptation can be realized is given.

6.3.2 Processing Incident and Alarm Descriptions

Provided that the FDH monitoring components within a node have detected a symptom for an erroneous state, this symptom is subsequently described using an information model and has to be made available for the purposes of the distributed self-healing control loop. This means that the belonging alarm/incident description has to be stored and disseminated across the network in question, such that the FDH agents inside the network nodes can react upon in an attempt to sustain quality of service and remove the root causes for the observed anomalies. Figure 6.9 describes this process from the perspective of the node within which the FDH monitoring components have detected the symptoms. These monitoring components are represented by the UML lifeline relating to the monitoring plug-ins.

There are two types of descriptions which might be generated and processed based on the specifics of a particular monitoring plug-in, namely alarm and incident description. Given that an alarm has been generated by the monitoring plug-in in question, then the belonging alarm description is pushed to the *Alarm Repository* of the local FDH node (*storeAlarmInformation* on Figure 6.9).

Subsequently, the local entities - represented through the *LE UML lifeline* [7] on Figure 6.9 - inside the FDH node are notified regarding the newly submitted alarm description (*notifyLocalEntities* on Figure 6.10). This includes, on one hand, the core FDH agents such as the local node FaultManAgent and SurvAgent, as well as other functional entities which might have registered at the Alarm Repository for receiving information of interest. That way these entities would be able to adapt their behavior according to the received notification. This is especially valid in clean slate type of networks (such as ANA [BJT+10]) where network protocols and functional blocks are designed from scratch and might be inherently developed with the awareness of a framework such as FDH. Having notified the local entities, the Alarm Repository proceeds with triggering the dissemination of the alarm notification throughout the network scope in question (e.g. a LAN or an OSPF area). This activity is represented by the *triggerAlarmDissemination* call on Figure 6.9. Subsequently, the EvDissAgent continues the flow with conveying the message (*disseminateAlarm*) regarding the newly generated alarm to relevant nodes across the FDH controlled network (represented by the *NET UML lifeline* [8] on Figure 6.9).

A similar procedure as the one above is performed when it comes to the processing of incident (failure/error/fault) descriptions. The lower part of Figure 6.9 describes the belonging steps including the storing of the incident information to the Local Incident Repository (*storeIncidentInformation*) , followed by the notification sent to the local functional entities (including the core FDH agents within a node), and finally triggering and conducting the dissemination of the incident description across the network scope in question (*triggerIncidentDissemination* and *disseminateIncident*). Thereby, the overall process is again initiated by the *Monitoring Plug-ins* operating within the FDH monitoring platform, and detecting the symptoms of faults from the perspective of a particular network node. The main difference between the initial processing of an incident and an alarm is given by the repository for storing and dispatching the information within an FDH node - the *Local Incident Repository* in case of an incident, and the *Alarms Repository* in case of an alarm.

The opposite aspect of alarm/incident processing is given by the case when an alarm/incident description is conveyed to an FDH node over the network. The FDH interaction flows that handle this aspect are depicted on Figure 6.10. The incident or alarm arrives at the local node (*reportIncidentOrAlarm* on Figure 6.10) and is first received by the *EvDissAgent*, which serves as a gateway between the FDH node components and the network as whole. In case the received message relates to an alarm, the belonging alarm description is stored (*storeAlarmInformation* on Figure 6.10) into the local *Alarms Repository* and forwarded to the core local FDH agents, i.e. FaultManAgent and SurvAgent, as well as to interested local functional entities. Subsequently, both core FDH agents can proceed with their local self-healing control loops.

In case the received message relates to an incident (failure/error/fault), the EvDissAgent has to differentiate whether the reported incident is related to the local node or not. In case the incident is not related to the local node, then it is stored in the *External Incident Repositories* of FDH (*storeIncidentInformation* on Figure 6.10) and the relevant local entities, including the FDH *FaultManAgent* and *SurvAgent*, are notified regarding the newly received external incident. Correspondingly, both key self-healing agents can involve the new external incident information in their belonging control loops.

Given that the received incident relates to the local node, i.e. a remote node asserts a malfunctioning of the local node, the incident description is first forwarded (*forwardIncidentInformation*)

[7] LE stands for Local Environment
[8] NET stands for Network

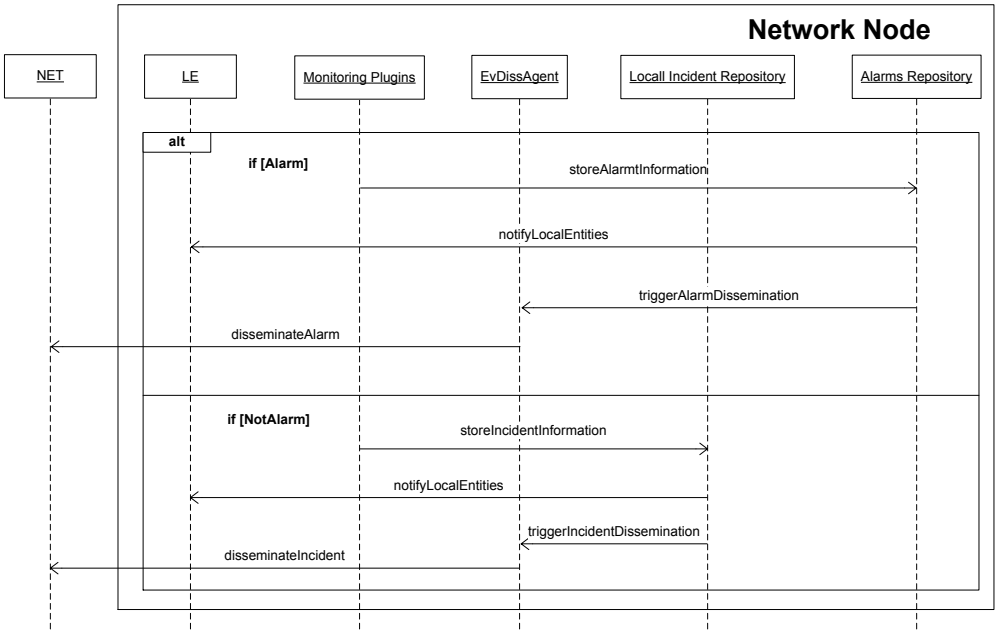

Figure 6.9: Processing Incidents and Alarms Originating from the Local Node

to the *Asserted Incident Assessment Functions* part of the FDH supporting components. The *AssIncFunction*-module executes some pre-configured policies, supplied by the network operations personnel, in order to verify the asserted incident locally (*assessAssertedIncident*). In case the asserted incident was confirmed, the belonging description is stored in the *Asserted Incidents Repository* of the FDH and all relevant local entities are notified. As a result, the FaultManAgent and the SurvAgent can use this incident information within their belonging control loops.

In case the asserted incident could not be confirmed, the *AssIncFunc* should trigger the dissemination of a recovery notification across the network (*triggerRecoveryNotificationDissemination* on Figure 6.10), given that the incident is obviously not present any more. This recovery notification is triggered over the *EvDissAgent* which notifies (*disseminateRecoveryNotification* on Figure 6.10) the FDH nodes in the network scope regarding the fact that the asserted incident is not present. As a result, the local *FaultManAgent* and *SurvAgent* will not involve the asserted incident description in their control loop executions, and the incident description will be removed from the FDH repositories across the network. This would hopefully lead to all FDH network nodes omitting the incident description in question within their local self-healing processes and improving the local self-healing operations.

Having shown how the FDH prepares and provides the set of alarms/incidents for the self-healing control loop, the next sections focus on how this information is utilized for self-healing by the FDH agents across the network.

6.3.3 Survivability Control Loops

Given that alarm and incident descriptions are handled and made available as described in the previous sections, the presentation of the dynamic aspects continues with the way a *SurvAgent*

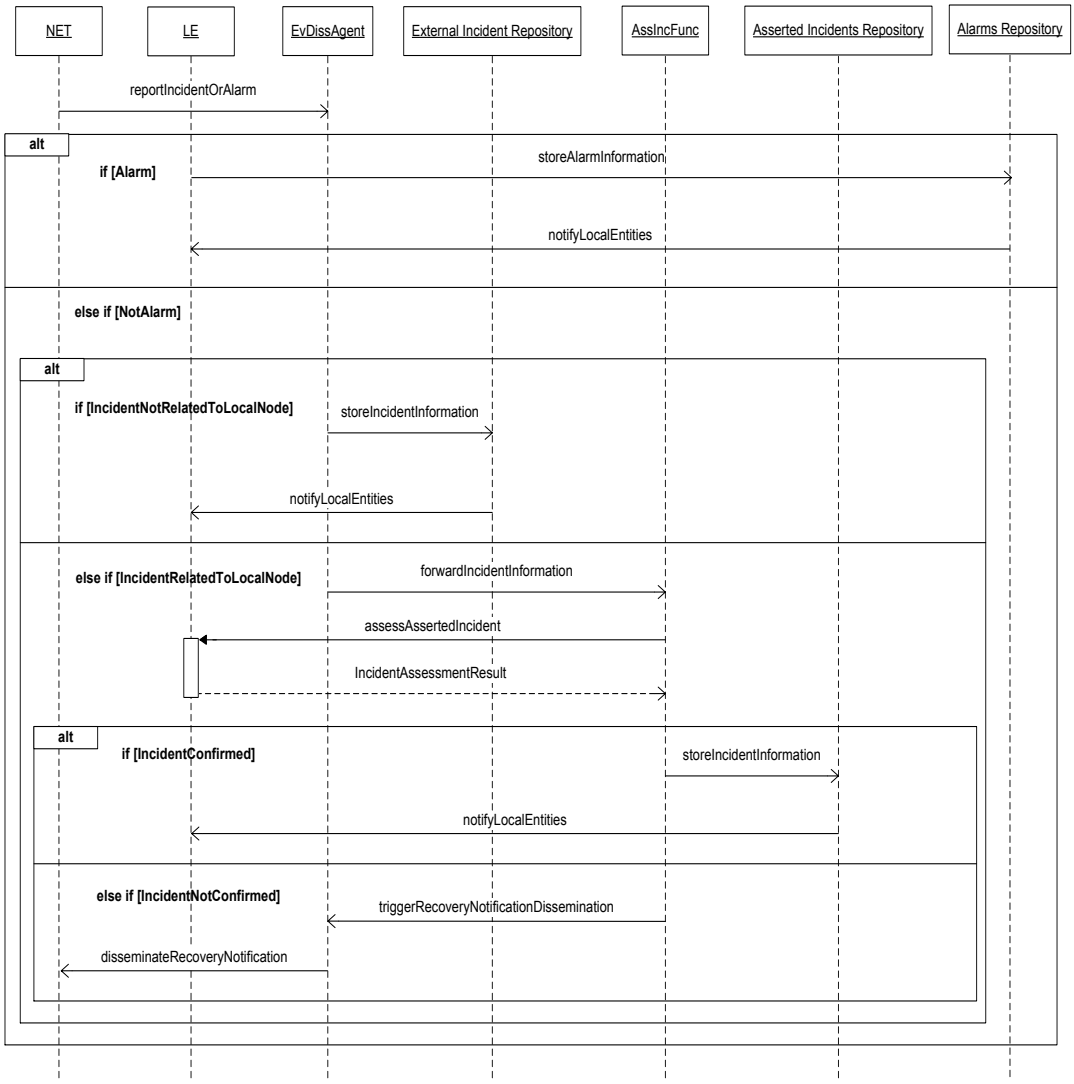

Figure 6.10: Processing Incidents and Alarms reported to the Local FDH Node from the Network

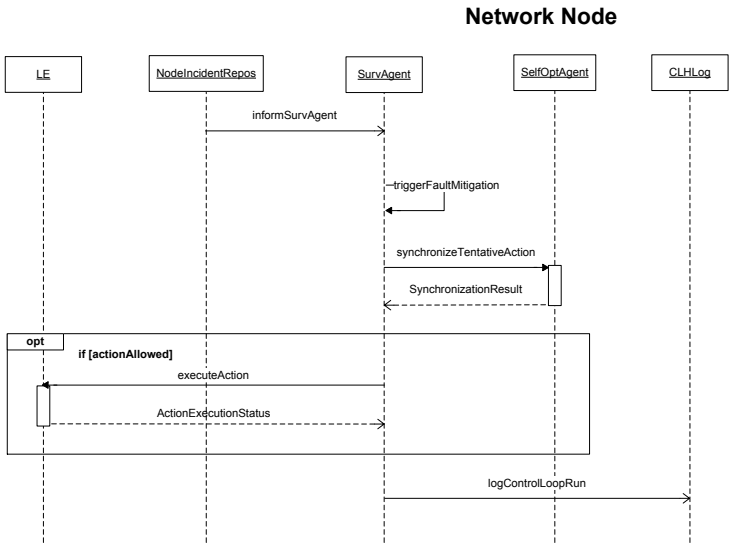

Figure 6.11: The Fault-Mitigation Control Loop within an FDH Node

within an FDH node consumes this information for its local control loop (see Figure 6.11). The SurvAgent is notified by the local FDH repositories (*informSurvAgent* on Figure 6.11) regarding a newly reported and locally stored incident or alarm. Based on this information, the local SurvAgent triggers a local Fault-Mitigation control loop (*triggerFaultMitigation* on the belonging sequence diagram). Correspondingly, the pre-configured *Fault-Mitigation* policies of the local SurvAgent are evaluated and an action prepared for execution, in order to mitigate the local effects of the reported erroneous state. Before running this tentative action, it needs to be approved for execution by the local *SelfOptAgent*, which is in charge of synchronizing the tentative actions of the various FDH control loops and optimizing the operation of the overall self-healing framework. This is achieved by the *synchronizeTentativeAction* call to the SelfOptAgent, and the belonging answer is given by the *SynchronizationResult* return value. Given that the tentative action was allowed by the SelfOptAgent, it is executed (*executeAction*) and the whole control loop is logged to the local control loop history log (*logControlLoopRun*).

In addition, Figure 6.12 presents the way monitoring information is consumed for the purpose of *Failure-Prediction* within the *SurvAgent* of an FDH node. The monitoring information is obtained by the *MonAgent* based on the monitoring requests which were submitted by the SurvAgent (*submitFailurePredictionMonitoringRequests* on Figure B.1) during the bootstrap phase of an FDH node. These monitoring requests result in KPI values periodically reported to the *Failure-Prediction Functions* of the SurvAgent. Hence, the *FPF* correlate the different monitored values (*correlateMonitoringInformation* on Figure 6.12) based on the node specific Failure-Prediction Models which were obtained (*getNodeSpecificFailurePredictionModels* on Figure B.1) during the bootstrap phase of the FDH node. Based on the node specific Failure-Prediction Models, the FPF might decide that an alarm needs to be generated since there is a high risk for a failure of a particular component, e.g. hardware component or an application server. Thereby, the alarm is used to first trigger local Fault-Mitigation. Subsequently, the belonging alarm description is stored in the local alarms repository (*storeAlarmInformationWithoutAutomaticallyNotifyingSurvAgent*) and

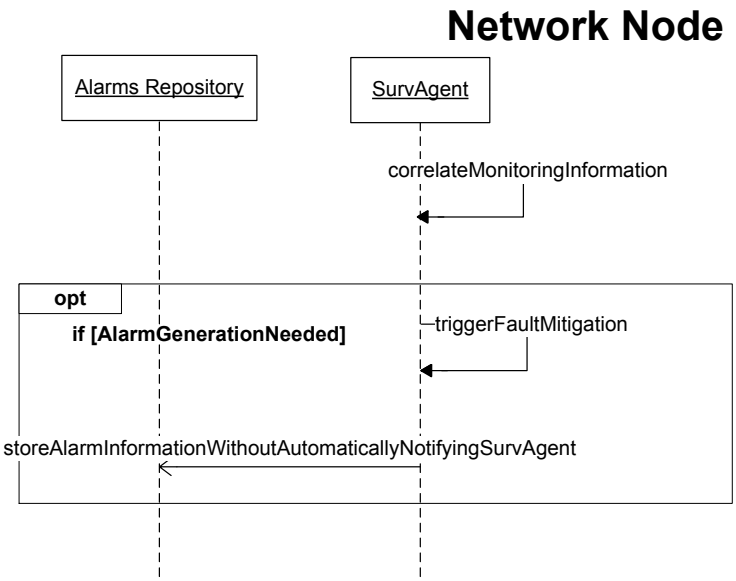

Figure 6.12: Failure-Prediction and Alarm Generation within the SurvAgent of an FDH Node

processed as described in the previous section, i.e. the local *FaultManAgent* is notified, relevant local entities are informed, and the alarm is disseminated by the *EvDissAgent* across the network, such that the self-healing control loops in other FDH nodes can react.

6.3.4 Fault-Management Control Loop

The last and most important FDH control loop to describe is given by the *FaultManAgent control loop*, which has the goal to remove faults in the long-term operation of the network. The interaction flows implemented by the FaultManAgent and its control loop in an FDH node are presented on Figure 6.13.

Given that the FaultManAgent has been notified every time a new alarm/incident description has been reported and stored locally - refer to Figure 6.9 and Figure 6.10 - the process of *Fault- Isolation* is triggered (*isolateFaults* on Figure 6.13) if one of the following conditions (which were already mentioned in section 6.2.1.3) is met:

1. Fault-Isolation is triggered every time a particular pre-defined number of incidents or alarms have been reported
2. Fault-Isolation may also get immediately initiated after a single alarm/incident event has been reported given that the event has a high (according to a pre-defined value) severity level, and
3. if only few (less than the pre-defined number of) incidents/alarms are reported and their severity is not causing the FIF functions to immediately trigger Fault-Isolation, then Fault-Diagnosis is triggered after a pre-defined time slice, which can be realized by the expiration

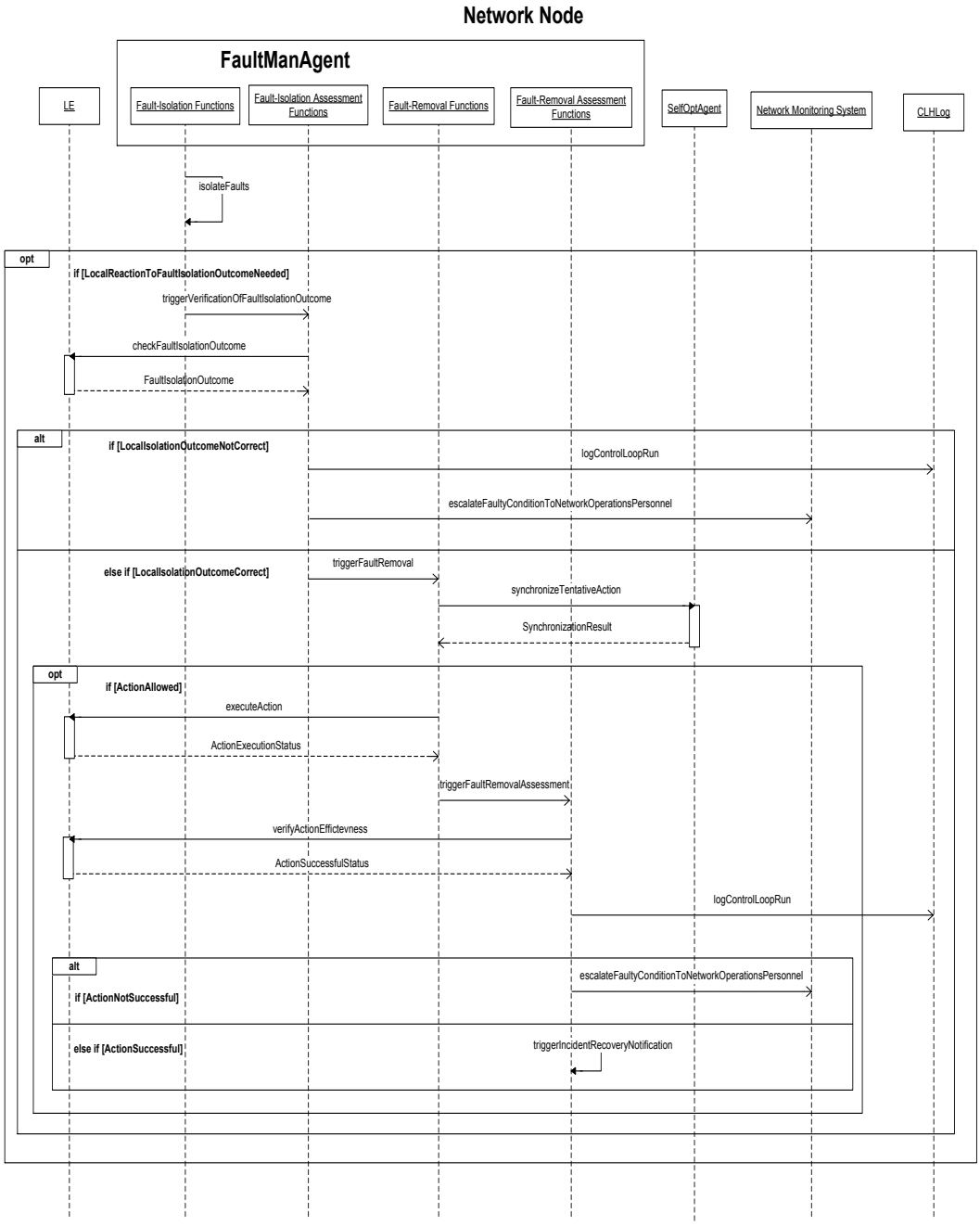

Figure 6.13: The FaultManAgent Control Loop within an FDH Node

of a timer

As previously mentioned, all the parameters required for evaluating the above conditions are supplied by the human experts tweaking the FDH processes.

The Fault-Isolation task uses the network-wide *Fault-Propagation Model* which is locally kept in the *FPMR repository*. After the Fault-Isolation results have been obtained, it needs to be checked on whether these results require a local reaction of the corresponding functions within the FaultManAgent. In case a local reaction is indeed required, the *FIAF* functions of the FaultManAgent must verify that indeed the faults, which were found based on the FPM, are activated (*triggerVerificationOfFaultIsolationOutcome* on Figure 6.13). This is done based on pre-configured policies, which lead to the required actions, in order to locally check the presence of a particular fault. The required action (e.g. probing or checking the run-time configuration of a functional entity) is executed on the local environment (*checkFaultIsolationOutcome* on Figure 6.13) and the belonging result is given by the return value *FaultIsolationOutcome*, which would keep a simple boolean value indicating whether the fault in question is indeed activated or not. In case the fault is not present, which would mean that the Fault-Isolation process was not correct, the situation is escalated to the network operations personnel (*escalateFaultyConditionToNetworkOperationsPersonnel* on Figure 6.13) and the control loop execution so far is logged to the Control Loop History Log (*logControlLoopRun* on Figure 6.13). Otherwise, if the isolated fault is indeed present, the Fault-Isolation result is passed to the *Fault-Removal Functions* of the FaultManAgent, in order to trigger the process of Fault-Removal (*triggerFaultRemoval* on Figure 6.13). In that context, the *FRF* operate based on pre-configured policies - supplied by the network administrators and downloaded from the Configuration Server during the bootstrap phase - that decide on the action(s) which is/are to be executed, in order to remove a set of faults. These action(s) would need to be synchronized over the SelfOptAgent (*synchronizeTentativeAction* and *SynchronizationResult* on Figure 6.13), in order to avoid contradictions with other FDH control loops and to assure the optimized operation of the FDH instances across the network. In case the tentative action was approved by the SelfOptAgent, it is executed on the local entities (*executeAction* on Figure 6.13) and the *Fault-Removal Assessment Functions* are triggered (*triggerFaultRemovalAssessment* on Figure 6.13), in order to check whether the fault(s) was/were indeed successfully removed (*verifyActionEffectiveness* on Figure 6.13). In case the set of faults was not successfully removed, an escalation of the situation to the network operations personnel is initiated (*escalateFaultyConditionToNetworkOperationsPersonnel* on Figure 6.13) after the control loop run has been recorded to the *Control Loop History Log*. In case of successful Fault-Removal, recovery notifications for all related incidents should be disseminated across the network (*triggerRecoveryNotification* on Figure 6.13), such that the incidents in question can be scheduled for deletion by the *Obsolete Event Handling Components* on the involved FDH nodes. In this line of thought, Figure 6.14 elaborates in details on the steps behind the *triggerRecoveryNotification* message/call on Figure 6.13.

Figure 6.14 illustrates the process which is initiated after a successful Fault-Removal, namely the process of disseminating recovery notifications for alarms/incidents related to the successfully removed faults (root causes for faulty conditions). The process is initiated by the *FRAF* functions, which firstly mark related alarm/incident descriptions as cleared (*clearRelatedIncidentDescriptions* on Figure 6.14) within the local node FDH repositories, such that these alarm/ incident descriptions are scheduled for deletion by the local *OEHC* component. Next, the *FRAF* module of the FaultManAgent triggers the *EvDissAgent* to disseminate a recovery notification (*triggerRecoveryNotificationDissemination* on Figure 6.14). As a result, the EvDissAgent notifies inter-

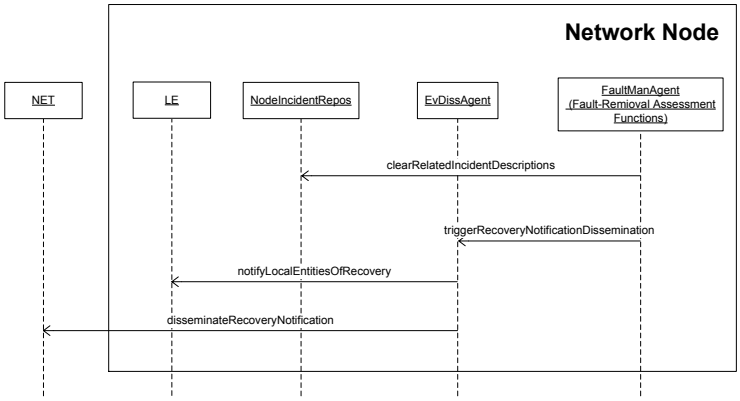

Figure 6.14: Dissemination of Recovery Notifications

ested local functional entities and subsequently disseminates the recovery notification across the network using the same mechanisms as in the case of alarm/incident dissemination. After the recovery notification has arrived on a remote FDH node, the belonging *EvDissAgent* accepts it (relate to Figure 6.15) and correspondingly invokes the *clearRelatedIncidentDescriptions* call on the alarm/incident repositories, in order to schedule the related alarm/ incident events for deletion by the belonging *OEHC* component. That way, the set of faults behind an erroneous state have been technically resolved and all belonging alarm/incident descriptions have been cleared from the FDH repositories across the network. This completes the **FaultManAgent Control Loop** within a node and the network as a whole.

6.4 Criteria for Escalating Faulty Conditions to the Network Operations Personnel

Having designed the distributed control loop on node and network level, the key aspect of the escalation of faulty conditions - not resolvable by means of the FDH framework - to the network operations personnel needs to be specified. Thereby, it is required to define what it means that a particular erroneous state is not resolvable by means of FDH. Based on this definition, the FDH control loops as well as the "Network Operations Personnel Interfaces" would be able to make decisions and notify the network administrators when required. Hence, a set of criteria are required which would allow the above mentioned FDH components and mechanisms to judge on whether a particular situation should be escalated or not. These criteria directly fulfill the corresponding **Req 22** from section 5.1.4. The following paragraphs discuss and define the concrete criteria for the FDH framework.

6.4.1 Increased FDH Overhead and Excessive FDH Activity

Given that the fundamental goal of a distributed system is to deliver its services in a qualitative and reliable manner, human intervention is needed whenever the FDH self-healing overhead is

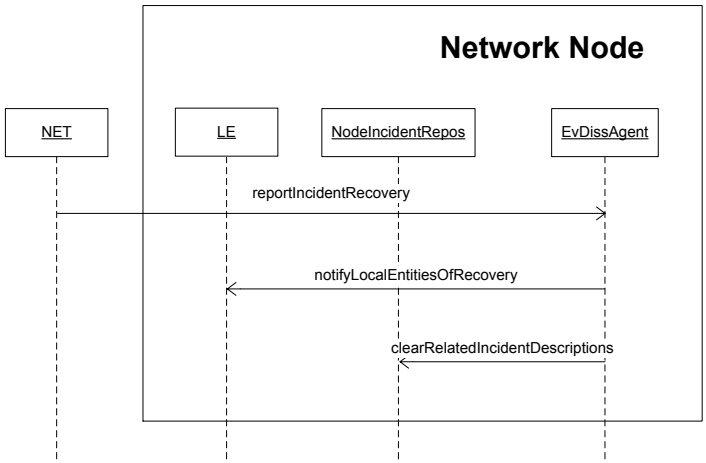

Figure 6.15: Receiving Recovery Notifications

increasing and is holding the network/system back from its fundamental task of forwarding traffic and delivering services, or when a problem is unresolvable by means of FDH based distributed self-healing. In the context of FDH, two cases have been identified that can indicate increased self-healing overhead. These are formulated in the following as criteria upon which the network operation personnel should be consulted, i.e. brought into the loop.

Criterion 1 - Incidents Overload: The FDH components should raise an alarm to the network operation personnel whenever too many (according to a pre-defined threshold) incidents are reported to the fault/error/failure/alarm repositories in the FDH nodes across the network.

Criterion 2 - Increased Control Loop Overhead: The FDH components should raise an alarm to the network operation personnel whenever the control loops for distributed self-healing are triggered too often (according to a pre-defined threshold).

6.4.2 Inability of FDH for Coping with an Erroneous State

As already mentioned, in addition to the above criteria the FDH mechanisms are expected to escalate and report the situation to the network administrators, in case there are increased indications that the FDH control loops are not able to cope with the challenges the network/system is exposed to. In the following, a number of such indications are listed, according to which the FDH control loops can be considered unable to resolve the issues the network/system encounters. Such issues would need human intervention and the involvement of the network administrators. Next, the indications are formulated as criteria upon which the particular situation should be escalated to the network operations personnel.

Criterion 3 - Unknown Incidents Reported: The FDH components should raise an alarm to the network operation personnel whenever an incident reported to the FDH node repositories is unknown within the Fault-Propagation Model of the network. In such a case, the processes of Fault-Isolation and consequently Fault-Removal cannot be triggered. Hence, the network is not able to react to

such an incident (self-heal), and the current situation should correspondingly get escalated to the network administrator.

Criterion 4 - Failed Fault-Isolation: The FDH components should raise an alarm to the network operation personnel whenever the Fault-Isolation process is not able to identify the root cause for the erroneous state of the network.

In the case of a probability based reasoning framework, the above criterion would mean that the probabilities calculated for the diverse potential root causes are too low and none of them can be diagnosed with a high value of certainty. Moreover, the FIAF functions of the FaultManAgent have the task to verify the correctness of the Fault-Isolation outcome. That is, the FIAF have to verify whether the identified root cause(s) (state(s)) are indeed activated, e.g. the configurations of an entity can be examined. If the suggested faults in question are not activated, then obviously the Fault- Isolation process has failed, and an escalation to the network operations personnel is required.

The next criterion relates again to the issue of a failed part of the FaultManAgent control loop, namely the Fault-Removal step.

Criterion 5 - Fault-Removal Failed: After the FRF module of the FaultManAgent has addressed an isolated fault (root cause), the FRAF part of the Fault-Management Agent would try to verify the elimination of the faulty condition by applying some probing/testing techniques. In case the FRAF functions assert that the faulty condition is still present, then an alarm should be raised to the network operation personnel, since it is likely that some of the FDH processes are operating in a wrong and inefficient manner.

The next criterion relates again to the potential inability of the FaultManAgent control loop to resolve a faulty condition. However, this criterion does not rely on particular FDH component to assess the outcome of a FaultManAgent control loop phase, but rather on observations regarding the alarm and incidents submitted to the FDH node repositories across the network.

Criterion 6 - Alarm/Incident Repeatedly Reported: The FDH components and mechanisms should raise an alarm to the network operation personnel whenever the same alarms/incidents are repeatedly reported (a number of times) to the alarm /incident repositories of the framework, although the corresponding control loops have already undertaken some actions to remove the assumed underlying root cause.

A situation as in the above criterion implies that either the process of Fault-Isolation or the process of Fault-Removal is failing to achieve its goal and the root cause for the corresponding faulty condition cannot be removed. Hence, an alarm describing the erroneous state should be conveyed to the human experts operating the network.

6.4.3 High Risk for a Component Outage

The Failure-Prediction Functions of the SurvAgent would produce estimations for the failure of diverse hardware components inside the network/system. An increased failure probability, based on observations related to the component in question, are vital information that can be used by the network operations personnel, in order to remove and substitute a hardware component that is likely to fail in the near future. Hence the following criterion is defined:

Criterion 7 - High Failure Probability: An alarm should be raised to the network operation personnel whenever the likelihood for a failure of a particular hardware component, estimated by the FPF

module of the SurvAgent agent, has reached a critical level.

All above criteria should be used to implement the interactions of the components of the FDH with the traditional Network Management/Monitoring System (NMS) as employed today in Network Operations Centers. The way these interactions are implemented within the FDH research prototype is presented in section 9 and section 10.7.

Chapter 7

Realization of the Self-Healing Function

The current chapter presents the conceptual and algorithmic aspects of the way the self-healing function of FDH is realized in the scope of the current thesis. The flow starts with describing the *FDH Monitoring Agent* and proceeds with the *alarm/incident/event sharing techniques* that were considered within the current work. Having described these aspects, and having that way laid the foundations for the sophisticated processes for handling alarms and incidents, the following sections proceed with describing the algorithms and models for realizing the critical *FaultManAgent* processes of *Fault-Isolation*, *Fault-Isolation Assessment*, *Fault-Removal*, *Fault-Removal Assessment*, as well as *Fault-Mitigation* and *Failure-Prediction* within the *FDH SurvAgent*. Thereby, the belonging requirements from chapter 5, which were already mapped to the corresponding components, are further refined and addressed on algorithmic level.

The description of the mechanisms implementing the FDH processes consists of various aspects. These include the specification of algorithms, structural concepts, meta-models and XML formats for capturing the required actions. Thereby, the algorithms are described using a pseudo code notation and their time and/or space complexity is discussed on. The time and space complexity discussions are conducted inline with the belonging requirements (**Req 23** and **Req 24**) from section 5.1.4. Specific scalability/performance/overhead measurements are presented in chapter 12, where the performance/overhead/scalability of the single FDH processes, as well as of the overall FDH distributed self-healing are analyzed in detail.

In order to demonstrate the applicability of the proposed algorithms, particular case studies are derived and presented. These case studies are later on considered as parts of some of the overall case studies presented in chapter 13, in which the whole distributed self-healing process is demonstrated and discussed on.

7.1 FDH Monitoring Agent

The *FDH Monitoring Agent*[1] is the FDH component that delivers basic monitoring services towards enabling the distributed self-healing realized by the FDH framework. Thereby, it plays the role of a Fault-Detection component, which provides input to the Fault-Management agents and the Survivability agents across the network. In addition the FDH Monitoring Agent can be requested (on-demand) to monitor and deliver information regarding different KPIs (reflecting net-

[1] The current section is an extended version of the work initially published in [TP12] and [LZM+10].

work/system performance and state), which can be further exploited by the FDH components if required.

The following subsections describe the different design principles, which were taken whilst realizing the monitoring functions required for the FDH framework. Thereby, these design principles are a further refinement of the high-level (reference model level) specification in section 6.2.1.1. Hence, the belonging monitoring requirements from chapter 5 are further addressed on technical level within the coming paragraphs.

It should be noted that the FDH Monitoring Agent is presented here based on its architectural models, which are very close to the implementation. The reason for this is that the realization of the FDH MonAgent is mainly architecturally driven, i.e. the internal structure of the agent and the interactions among the sub-modules determines its operation, whilst there are hardly any significant algorithms of vital importance. This is a major difference to the other aspects of the self-healing function (and correspondingly the other sections of this chapter), which are presented with a focus on algorithmic aspects, whilst the implementation and configuration models are addressed in the sections relating to the prototype implementation and to the network operations personnel view on FDH.

7.1.1 Extensible Monitoring Functions Design

The FDH Monitoring Agent must be extensible and configurable to deal with different types of requests for Fault-Detection and KPI's monitoring. In order to achieve that, it is needed to come up with structures which would allow the extensibility of the monitoring platform depending on the self-healing needs of the particular network. The key element within this concept is the one of a monitoring plug-in - be it a Traffic Monitoring Plug-in or a Node Monitoring Plug-in - as specified in section 6.2.1.1 and Figure 6.3. Therefore the FDH MonAgent is designed, and its specification is further refined, as to be extensible by plug-ins which would be doing the job by combining different monitoring tools and accessing belonging monitoring interfaces (e.g. SNMP MIBs or CLIs). Indeed, the interfaces between the various FDH MonAgent modules (e.g. monitoring plug-ins or *Data Aggregation and Correlation Framework* of the FDH MonAgent) must be specified in way that they can serve the purpose of being accessed by plug-in developers, who would be normally be constituted by network/system administrators implementing monitoring functions required for a particular network/system. In the following, the modeling concept of *UML interface diagrams* [Bel04] is used to capture the APIs required and made available for the deployment of monitoring plug-ins enabling FDH based self-healing.

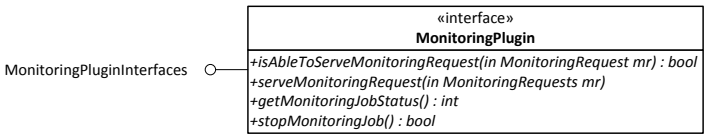

Figure 7.1: Monitoring Plug-in Interfaces

The interfaces which are required for the implementation of an FDH monitoring plug-in are given in Figure 7.1. These interfaces are to be seen as abstract functions, which are later to be implemented and provided by each plug-in, such that it can be integrated within the FDH MonAgent

flows. In the scope of the Java based implementation presented later on this thesis, these interfaces are realized as Java *interfaces*, which fix the way a monitoring plug-in is addressed by the rest of the MonAgent components - after having been loaded and initialized within the FDH MonAgent platform.

Coming back to the interfaces described in Figure 7.1, a monitoring plug-in should be able to recognize whether a monitoring request - submitted by one of the other FDH agents (see Figure B.2- can be processed by the monitoring plug-in. The logic for the identification of such relevant monitoring requests should be implemented within a specific method specified as *isAbleToServe-MonitoringRequest(in MonitoringReques mr): bool* in Figure 7.1. This method is invoked by the *Orchestration Engine* of the FDH MonAgent after a monitoring request has been sent/submitted by another FDH agent. Thereby, the Orchestration Engine forwards the request to each available monitoring plug-in. That means, that the *isAbleToServeMonitoringRequest* method of each plug-in is invoked with the corresponding monitoring request as parameter, which is in turn analyzed if it is suitable to process by the current plug-in, and correspondingly a boolean value is returned that determines whether the request can be served by that plug-in or not. Provided that a monitoring request has matched a particular monitoring plug-in using the above interface, the monitoring request should be subsequently passed to the plug-in for further processing. This is realized via the *serveMonitoringRequest (in MonitoringRequest mr)* interface. The reason for separating the matching of a request from the submitting of the request is constituted by the idea to give the MonAgent the freedom to implement a logic for selecting a monitoring plug-in, in case multiple plug-ins match a particular request. Finally, a monitoring plug-in should be able to report the status of the running monitoring job, which is to be realized via the *getMonitoringJobStatus(): int* interface. In that case the status is encoded as an integer value, which can reflect different states of the monitoring process. Through the above specified interfaces, it should be guaranteed that the FDH MonAgent platform can address and interact with plug-ins, which implement monitoring functions specific to the needs of the network/system in question. Furthermore, it is required to define interfaces which allow the monitoring plug-ins to interact with other modules within the FDH MonAgent. These interfaces are described in the following paragraphs. Finally, it should be possible to stop a monitoring job, which is realized over the *stopMonitoringJob() : bool* interface to be implemented by each plug-in.

After having captured a particular KPI value, the monitoring plug-in pushes it to the *Data Aggregation and Correlation Framework* of FDH (see Figure 6.8 in section 6.3.1). The *DataAggrCorrFram* is subsequently responsible for selecting the corresponding data aggregation/correlation function and forwarding the resulting KPI value to the *Orchestration Engine* of the FDH MonAgent (refer to Figure 6.8). In order to push monitored KPI values to the DataAggrCorrFram, a monitoring plug-in uses the *pushDataForCorrelation (in KPI k, in int CorrelationAndAggregationMethod)* interface captured by the UML interface diagram in Figure 7.2. Thereby, an object containing the measured KPI value is passed, as well as an integer that is meant to contain a code representing the requested aggregation/correlation method - e.g. average, exponential average, or median. The belonging functionality for the correlation must be offered as part of the monitoring platform and the provided *Data Aggregation and Correlation Framework*.

Besides, according to the interaction flows described in Figure B.2 and Figure 6.8, there are particular points at which the evaluation of some policies is required - e.g. admission, adaptation or notification policies regarding a particular plug-in, its monitoring behavior or the KPI values it has monitored. These policies are indeed required to follow a particular format, which would enable them to be integrated and utilized at the corresponding places within the interaction sequences. The way these policies were realized within the FDH prototype is described in the coming sec-

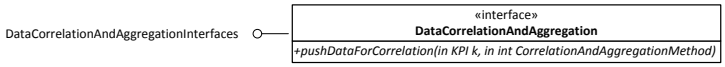

Figure 7.2: Data Correlation and Aggregation Interfaces

tions.

7.1.2 Meta-Models for the MonAgent

The following subsections focus on the meta-models, which are needed for the operation of an FDH MonAgent within a network node.These include the different policies that are needed for the internal processes of the monitoring platform, as well as the meta-model for defining and expressing monitoring requests.

7.1.2.1 FDH MonAgent Policies

The operation of the FDH MonAgent inside a network device requires various types of policies that facilitate the FDH MonAgent to take key decisions regarding the monitoring process. The relevant policies were initially captured in Figure 6.3 in section 6.2.1.1.

These policies include the *Admission Policies* of an FDH MonAgent that determine whether a particular monitoring request - coming either from the FaultManAgent or the SurvAgent - should be accepted and fulfilled based on various constraints, which may include for example aspects of resource (memory, CPU utilization, etc.) consumption, or the number of monitoring jobs already in place. Other policy sets, which need to be provided by the network operations personnel, are constituted by the *Notification Policies* and the *Adaptation Policies* for an FDH MonAgent within a node.

The *Notification Policies* contain the logic for decisions, on whether a particular measurement should be communicated to the agent, which has submitted the belonging monitoring request. This decision would normally be based on particular thresholds that determine if a value, which was measured for a KPI, constitutes an abnormal situation or not. Furthermore, the *Adaptation Policies* are invoked at a particular point of the monitoring information processing (specified in Figure 6.8 as the point after the notification/decision w.r.t. the monitored KPI values) and trigger steps as to adapt the belonging monitoring job/plug-in, depending on key aspects within the last monitoring period. Parameters of a monitoring job/plug-in, which may require adaptation, are given by the *duration of a monitoring period*, the *granularity/rate* of the monitoring job invocation, the *storage policy for measured values* - which KPI measurements require to be stored in a local database and which can be dismissed immediately, the *measurement aggregation parameters*, such as *number of measurements* or *type of aggregation function*, e.g. median [She07], weighted average [She07], or simple arithmetic mean [She07]. The adaptations would take into account aspects such as resource consumption issues (e.g. memory consumption, CPU utilization, etc.), the dynamics of the measured KPIs - for example how strongly do the measured values vary, and further aspects which would depend on the particular monitoring job and implementing monitoring plug-in.

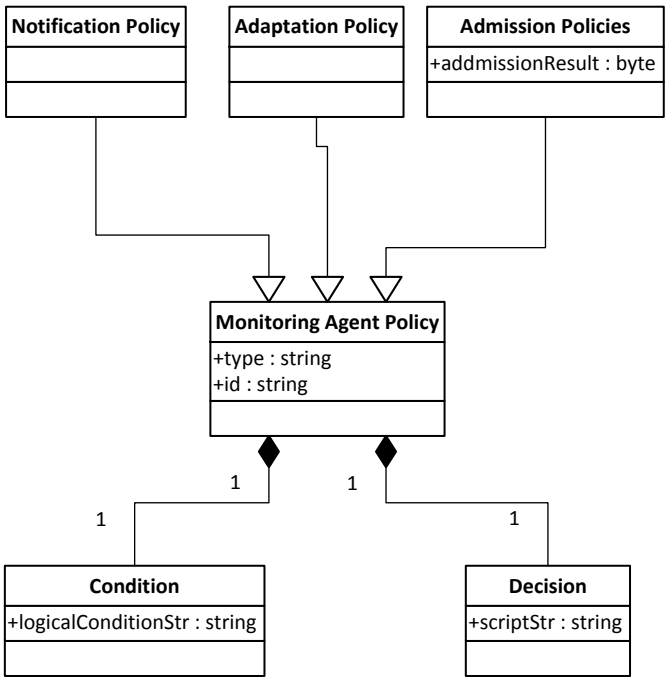

Figure 7.3: Meta-Model for the Policies required by the FDH Monitoring Agent in a Node

The meta-model for the different types of policies required by an FDH MonAgent within a node is illustrated in Figure 7.3. It builds on an abstract Monitoring Agent Policy depicted in the center of Figure 7.3. This abstract policy contains the fields *id* and *type*, which are meant to identify it and to provide a short description, such as type of the policy or a short combination of keywords capturing key aspects of the policy (e.g. RTT_MONITORING_NOTIFICATION). The abstract Monitoring Agent Policy is further composed of a *Condition* and a *Decision*, which are both represented by strings. The *Condition* is meant to be a logical expression which can be evaluated and the *Decision* string is meant to represent the action which is executed based on the outcome of the condition [2]. Finally, the Monitoring Agent Policy is specialized (in terms of being the basic class) by the particular policies required for the different steps of the FDH MonAgent operations. Thereby, the *Admission Policies* bring in a new attribute in particular, which is given by the admission result that is used by the Orchestration Engine of the MonAgent, in order to dismiss or proceed with a particular monitoring request.

A final remark on the presented policies meta-model: It is considered that the monitoring policies are node-specific and should be separately provisioned for the FDH MonAgent for each device in a distributed network/system.

[2] In the current prototype, the *Condition* and the *Decision* part are implemented based on the Java BeanShell [Bea16] library, which allows for dynamically executing pieces of Java-like code. More details on the implementation including examples of policies are given in chapter 10.

7.1.2.2 Monitoring Requests

The monitoring requests are the second key aspect that makes the operation of an FDH MonAgent within a node. These monitoring requests are downloaded by the key FDH agents within a node - the FaultManAgent and the SurvAgent - and submitted in turn to the local FDH MonAgent, in order to initiate required monitoring jobs. That is, the decision on when to submit a monitoring request to the MonAgent, and that way initiate a monitoring job, is left to the operational logic of the other FDH agents. Therefore, the download of the monitoring requests from the centralized FDH Configuration Server is realized by each of the key FDH agents (FaultManAgent and SurvAgent) for its own set of monitoring requests The interactions required to accomplish these steps are depicted in Figure B.1.

The meta-model for FDH monitoring requests is provided in Figure 7.4. It is inspired by the Network Resource Model of 3GPP [3GP13], which is taken as a pattern regarding how to describe a particular network resource that needs to be monitored. This pattern includes the aspect of a *Subnetwork*, which contains a number of nodes that can be either a *Host* - i.e. a machine connected to the network providing services or applications, or a *Router* - a forwarding node on network layer. Both types of network nodes possess interfaces to the network - *DeviceInterface* with belonging IP addresses as well as a device identification string, e.g. *eth0*. In addition, various node-specific aspects can be monitored within a device which is given by the *NodeMonitoringAspects* abstraction.

The concept of a *Subnetwork* comprising the above aspects is the key item within an abstracted *MonitoringJob* that also includes the attributes of *granularityPeriod*, i.e. the pause between two periodical invocations of the monitoring job, and the total number of monitoring periods (*numberOfMonitoringPeriods*), i.e. the number of times the monitoring job will be periodically started. The concept of a *MonitoringJob* together with the above described *Subnetwork* and comprising aspects round up the abstract part of the meta-model, which contains abstract classes that must be specialized (depicted through the UML generalizations) for the purpose of describing concrete monitoring jobs as well as concrete resources to be monitored.

Focusing first on the resources to be monitored in Figure 7.4: the *DeviceInterface* can be directly included in the monitoring jobs as a resource to be monitored, whilst the *NodeMonitoringAspects* are specialized by further model elements describing specific node KPIs to be observed, such as open *FileDescriptors* per process, *MemoryUtilisation* and *CPUUtilization* per process. Naturally, it is possible to add further node KPIs beyond those which were implemented for the scenarios in this thesis.

Moving on to presenting the specific jobs, various monitoring jobs have been specified for the purpose of the scenarios in this thesis. All these jobs are based on the abstract *MonitoringJob* presented above and contain automatically the attributes of a *granularityPeriod* and *numberOfMonitoringPeriods*. In addition, many of these jobs contain threshold attributes or IP addresses that specify the network interfaces/devices towards which particular probes must be sent (e.g. in case of the RTT monitoring job). Furthermore, many of these jobs are meant to detect incidents or generate alarms and directly submit those to the repositories of FDH. In these cases, attributes are put in place that specify the interface to which the incident/alarm information is to be sent [3]. As in the case of the KPIs to monitor, the meta-model can be extended by further monitoring jobs depending on the requirements of a particular network/system and the addressed scenarios.

[3] In the concrete implementation, these interfaces are given by Unix Domain Sockets paths where the information is sent to.

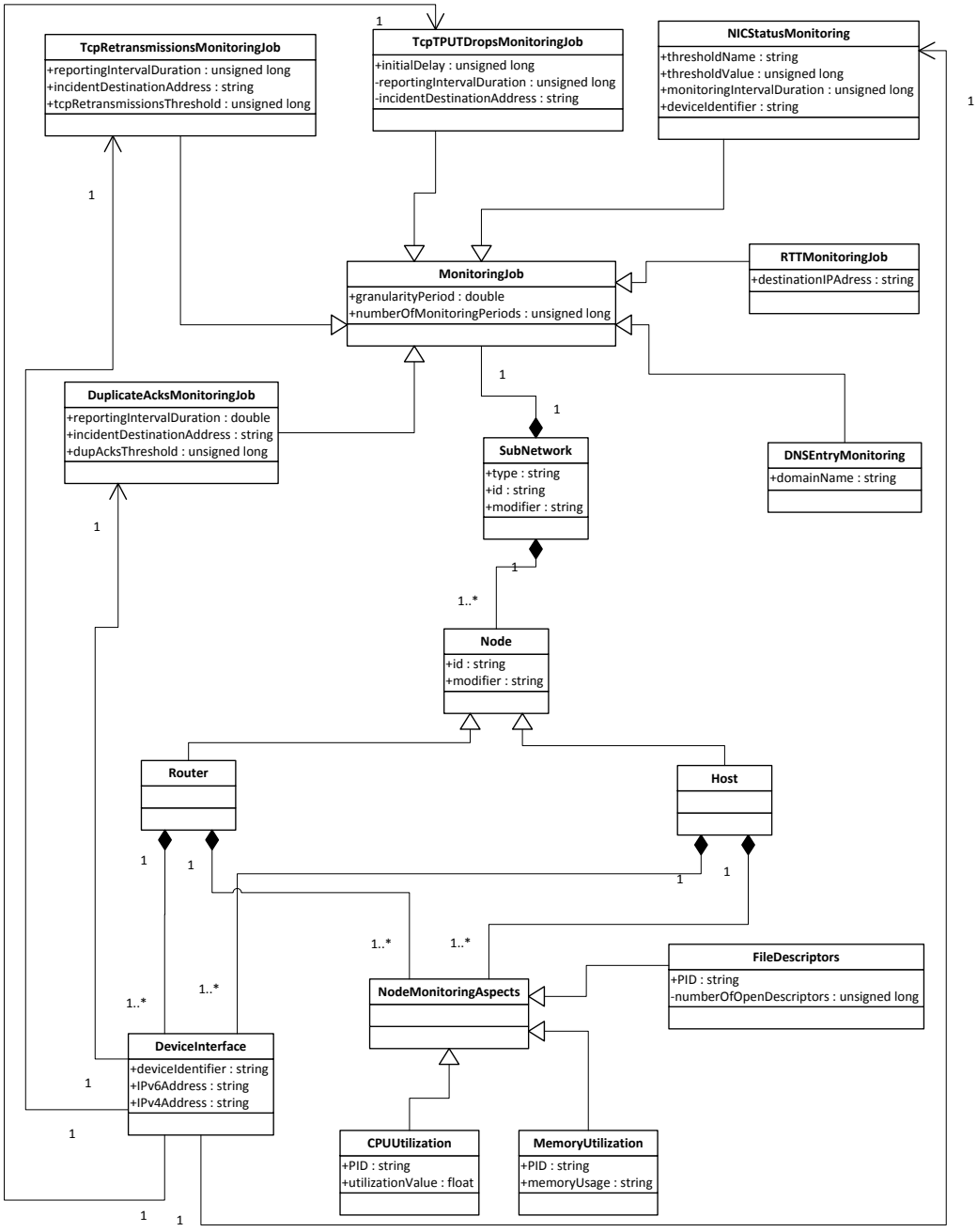

Figure 7.4: Meta-Model for Monitoring Requests

Further details regarding specific implementation aspects - based on the meta-models and FDH MonAgent specifications presented hitherto - are described in section 10.2. In parallel, the following section demonstrates the application of the above monitoring related concepts to real network problems, which are relevant for the case studies of the current thesis.

7.1.3 Example Applications

The current section presents how the FDH MonAgent was instrumented to monitor for IP Black Holes [Lah00] and for Ethernet Duplex Mismatches [SC05]. Thereby, the FDH MonAgent is configured as to detect symptoms for the erroneous states in question. Hence, the MonAgent in that context takes care of orchestrating monitoring services that realize Fault-Detection based on requests submitted by the FDH FaultManAgent.

The meta-model for specifying these requests was defined in section 7.1.2.2. Furthermore, an example for such an XML request based on the above meta-model is given in section 10.2, which elucidates various configuration aspects of the FDH framework.

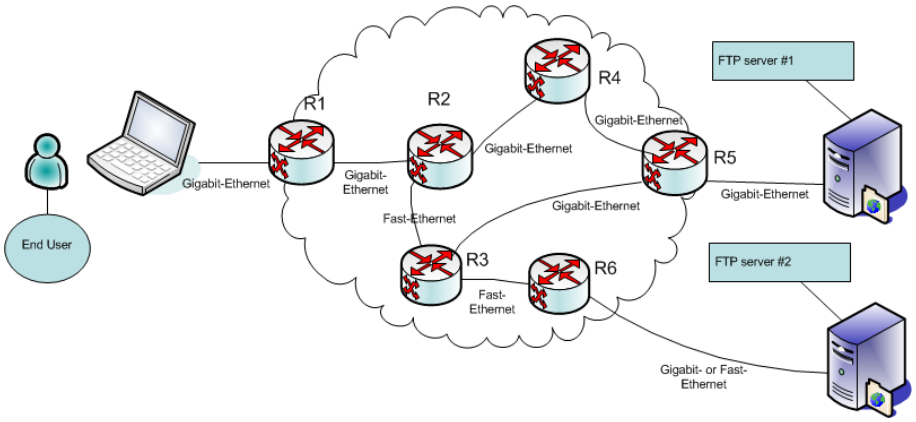

Figure 7.5: Reference Network on which the Case Study is based

The case study was deployed on the reference testbed depicted in Figure 7.5. This testbed reflects a network which is used to play case studies for the algorithms and techniques which were designed for the various components of FDH. Thereby, this sub-network [4] is constituted by a part of the overall testbed presented in section.

7.1.3.1 Ethernet Duplex Mismatch

The problem of Ethernet Duplex mismatch appears especially in Fast-Ethernet (IEEE 802.3) networks when the auto- negotiation procedure between the Network Interface Cards (NICs) on a link has failed. This auto-negotiation procedure is meant to ease the setting up of an IEEE 802.3 based network. However, there are particular situations which were reviewed in [SC05] and can lead to a mismatch in the duplex mode (e.g. full vs half-duplex mode) assumed on the Fast-Ethernet NICs on both ends of a link.

According to [SC05], a duplex mismatch on an IEEE 802.3 link can occur in the following setups:

[4] Sub-network is not meant in the sense of an IP sub-network, but in the sense of the given network being part of the larger testbed

1. *"One card is hard-coded to use full-duplex and the other is set to auto- negotiate: the hard-coded side will not participate in negotiation and the auto-negotiating side will use its half-duplex default setting;*
2. *The two cards are hard-coded to use different duplex modes;*
3. *Both cards are set to auto-negotiate, but one or both of them handles auto-negotiation poorly " ... "note that, in this case, the problem can occur sporadically and rebooting or resetting the interface on either side could clear the condition."* [SC05]

Such a situation would lead to frames being sent out on the link by the NICs without consideration regarding whether the interface on the other end is simultaneously issuing Ethernet frames or not. Obviously, this would lead to a large number of lost IP packets and has the potential to severely cripple the performance of a network.

Based on the information provided in [SC05], a number of plug-ins for the detection of symptoms of duplex mismatch were developed. Thereby, small programs were implemented which use various available monitoring tools in order to obtain required KPI values. These small programs were in turn wrapped into Java based FDH MonAgent plug-ins that implement the interfaces described in section 7.1.1 above.

Following plug-ins for detecting symptoms of IEEE 802.3 duplex mismatch according to [SC05] were provided:

- *NicStatusPlugin* - this plug-in is responsible for obtaining various KPIs during the operation of a Network Interface Card. Thereby, it utilizes the *ethtool* [ETH05] and the *ifconfig* [IFC00] tools to obtain performance indicators such as the TX/RX counters of a NIC, as well as the number of dropped packets at a particular Network Interface Card. An increased number of dropped packets is a symptom that might be occurring due to an IEEE 802.3 duplex mismatch. The identification of whether the root cause is indeed given by such a duplex mismatch is the task of the Fault-Isolation process, which runs on a Fault-Propagation Model capturing the various possible chains of fault-propagation within a network. The requests for this plug-in are based on the *NICStatusMonitoring* part of the meta-model in Figure 7.4.
- *TcpRetransmissionsPlugin* - the *TcpRetransmissionsPlugin* uses available packet capturing tools (*tcpdump* [TCP15] within the current prototype), in order to track TCP packet flows over an interface and identify an increased number of TCP retransmissions. An increased number of TCP retransmissions is another indication that potentially an IEEE 802.3 duplex mismatch has been activated on an Ethernet link. The requests for this plug-in are based on the *TcpRetransmissionsMonitoringJob* part of the meta-model in Figure 7.4.
- *DuplicateAcksPlugin* - *duplicate acks* constitute a TCP protocol feature that enables a receiver to notify a sender that packets are arriving in a messed up order and that obviously packets in between have been lost. The corresponding plug-in for detecting this anomaly (whenever occurred) is developed as to use packet capturing tools (*tcpdump* [TCP15] within the current prototype) and to analyze TCP traffic on an interface, in order to detect increased appearances of *duplicate acks*. Such increased appearances of *duplicate acks* are indicative for packet losses and might eventually mean that an IEEE 802.3 duplex mismatch has occurred on an Ethernet link. The task of correlating the various symptoms, and identifying whether indeed a duplex mismatch is the reason for the crippled network performance, is

fulfilled by the Fault-Isolation process based on a Fault-Propagation Model capturing the potential fault-propagation chains within the network in question. The requests for this plug-in are based on the *DuplicateAcksMonitoringJob* part of the meta-model in Figure 7.4.

The above presented plug-ins were conceptualized and realized to monitor their respective KPIs for a Network Interface Card. Hence, it is expected that depending on the characteristics of a network, these plug-ins would be activated to monitor some key NICs on selected nodes. Coming back to the reference network for the monitoring case studies (Figure 7.5): The requests for detecting *Ethernet Duplex Mismatches* were served on the routers *R2*, *R3*, and *R6* in the testbed on Figure 7.5. These routers have Fast-Ethernet NICs attached, which potentially can have the problem of a duplex mismatch as described in [SC05]. The above described plug-ins realize the Fault-Detection with respect to the identified symptoms and would be submitting their observations to the FDH repositories residing in the routers, as well as to the corresponding FDH FaultManAgent and SurvAgent within a node.

7.1.3.2 IPv6 Black Holes

The second Fault-Detection case study deals with a specific version of the problem of Black Holes in IP networks and especially in IPv6 networks, which have undergone a transitioning from an IPv4 based network configuration. The particular type of IP Black Hole that was addressed within the case study is described in detail in [Lah00].

In general, the term Black Hole refers to a router that silently drops packets without sending any notification to the involved communication participants, in particular sender or receiver. Thus, it is difficult for the sender to regulate the settings of the flow it generates. That way, big amounts of traffic can get lost causing the network to fail delivering its services. [Lah00] describes the above mentioned specific type of a Black Hole phenomenon that can potentially occur in IPv6 networks, which have inherited their firewall configurations from a previous IPv4 deployment, e.g. due to IPv4-to-IPv6 transition. The misconfiguration of the firewall is constituted by the suppression of ICMPv6 "Packet too big" messages. These messages are meant to notify the end systems in case the PMTU (Path Maximum Transmission Unit) changes during the lifetime of a flow (e.g. because of a link failure and automatic traffic rerouting), such that the sender can adjust the size of the packets it sends out.

The latter situation can lead to extensive packet loss in the scope of IPv6, because in IPv6 networks packets do not get fragmented on intermediate router nodes, in order to reduce the performance overhead in the forwarding plane. However, in IPv4 networks, it was a common practice to suppress many different types of ICMP messages due to the fact that implementation bugs have made them an easy vehicle for an attacker to flood the network and achieve service unavailability. In contrast, in an IPv6 network, the suppression of ICMP in firewalls is a critical issue that should be carefully handled according to the belonging IETF RFC, e.g. RFC4942 [DKS07].

On the network in Figure 7.5, R1 would be a router that can potentially cause such a Black Hole problem, provided that R1 has different types of links attached - a Gigabit-Ethernet link that can carry up to 9000 bytes (e.g. with Jumbo Frames), and a Fast-Ethernet link with an MTU of maximum 1500 bytes. In a situation where a path has been established that contains only Gigabit-Ethernet links - such as *End User* $\leftrightarrow$ *R1* $\leftrightarrow$ *R2* $\leftrightarrow$ *R4* $\leftrightarrow$ *R5* $\leftrightarrow$ *FTP server #1* - a Path MTU (PMTU) of more than 1500 bytes would be established for the flows on this path. In case of a failure on the link R2 $\leftrightarrow$ R4, the dynamic routing protocol (e.g. OSPFv3 or RIP-ng) on *R2* would

take care of rerouting the above flows over the *R2* ↔ *R3* link, which would lead to a decrease in the PMTU that would be then set down to the 1500 bytes MTU of the Fast Ethernet (FE) link (*R2* ↔ *R3*). This would lead to R2 failing to forward packets over the *R2* ↔ *R3* link because of the link being unable to carry the size of the packets which are sent out based on the PMTU established over the Gigabit-Ethernet path (*End User* ↔ *R1* ↔ *R2* ↔ *R4* ↔ *R5* ↔ *FTP server #1*), which would be larger than 1500 bytes. Normally, *R2* would notify the sending device (*End User* on Figure 7.5) via an ICMPv6 "Packet too big" message to reduce the size of the packets it sends out. However, the presumed firewall misconfiguration on *R2* (due to the IPv4-IPv6 transitioning) would suppress the notification via the ICMPv6 "Packet too big" message, which would imply that the end user system continues sending out packets that are silently dropped at *R2* leading to a Black Hole at this router.

The symptoms of such a Black Hole are partially described in [Lah00]. In the current study, monitoring sensors were instrumented, which were observing TCP flows on *R1* order to detect duplicate acks, increased numbers of retransmissions or sudden drops in TCP traffic that are indications for Black Hole problems. Hence, following monitoring plug-ins were conceptualized and deployed [5] within the FDH MonAgent on *R1*:

- *TcpRetransmissionsPlugin* - as described above, the *TcpRetransmissionsPlugin* utilizes the *tcpdump* [TCP15] packet sniffer for the purpose of capturing TCP flows and detecting increased numbers of TCP retransmissions. This anomaly constitutes a symptom for an IP Black Hole besides being also an observable anomaly in the course of IEEE 802.3 duplex mismatch. Once detected, the observation is submitted to the other FDH agents - within a node and eventually across the network - where it is used within the Fault-Isolation process, in order to identify the correct underlying root cause - be it an IEEE 802.3 duplex mismatch or an IPv6 Black Hole.The requests for this plug-in are based on the *TcpRetransmissionsMonitoringJob* part of the meta-model in Figure 7.4.

- *DuplicateAcksPlugin* - as described in the previous section, the *DuplicateAcksPlugin* uses the *tcpdump* [TCP15] packet capturing tool, in order to analyze TCP flows passing through *R1* and to detect increased number of *duplicate acks*. The symptom of *duplicate acks* stands for a flow where packets arrive in the wrong order and the receiver notifies the sender to re-send missing packets. Thus, *duplicate acks* might also occasionally appear in case of an IP Black Hole activation, in cases when large packets fail to get through whilst small ones succeed. Given that such an increased number of *duplicate acks* has been detected, the *DuplicateAcksPlugin* submits an incident to the other FDH agents in the local node and subsequently across the network (after the dissemination process has been accomplished), where the *duplicate acks* incident is used in the Fault-Isolation process towards the identification of the underlying root cause (e.g. a Black Hole occurrence or an IEEE 802.3 duplex mismatch).The requests for this plug-in are based on the *DuplicateAcksMonitoringJob* part of the meta-model in Figure 7.4.

- *TcpTputDropsPlugin* - another anomaly which is observed in case of an IPv6 Black Hole occurrence is constituted by sudden drops in the throughput of TCP flows passing through a router between the sender and the Black Hole router - in the current case this router is given by *R1*. These sudden drops are detected by the *TcpTputDropsPlugin* orchestrated by the FDH MonAgent on *R1*. Thereby, the *TcpTputDropsPlugin* uses the *tcpdump* [TCP15] packet capturing tool, in order to track TCP flows passing through *R1* and to search for

[5] Implementation details are provided in section 10.2

sudden drops in the TCP throughput for the captured traffic. Given that such an anomaly has been detected, the TcpTputDropsPlugin submits a corresponding incident description to the FDH platform. This description is later used in the process of Fault-Isolation within the FDH nodes. The requests for this plug-in are based on the *TcpTPUTDropsMonitoringJob* part of the meta-model in Figure 7.4.

The above listed plug-ins utilize theMonAgent plug-in interface specifications[6] - refer to the *MonitoringPlugin* interface in Figure 7.1.These plug-ins are deployed and orchestrated by the FDH MonAgen on *R1*. In order to test the plug-ins, the IPv6 Black Hole problem was introduced, i.e. the R1 firewall was configured to suppress ICMPv6 "Packet too big" messages and the *R2* ↔ *R4* link was brought down such that the running traffic between *End User* and *FTP server #1* had to be re-routed over the Fast-Ethernet link with the lower MTU (i.e. the *R2* ↔ *R3* link). Thereby, the monitoring plug-ins were detecting the above symptoms on *R1* and pushing incident descriptions to the FDH agents. Indeed, it was observed that the sudden throughput drops were always detected whenever the IPv6 Black Hole on *R1* was activated, whilst the *duplicate acks* and the TCP retransmissions were only sporadically observed. This can be explained by the somehow stochastic nature of the monitoring plug-ins given that they periodically sample traffic and subsequently analyze it. Hence, it might happen that the specific occurrence of retransmission/"duplicate acks" anomalies (before the TCP flow stalls) is missed by the sampling processes.

The above described case studies demonstrate the feasibility and applicability of the FDH MonAgent concepts and architecture for the purpose of Fault-Detection for distributed self-healing in IP networks. Thereby, it was shown how the FDH MonAgent can be used to monitor and detect symptoms of IEEE 802.3 duplex mismatch and of IPv6 Black Holes resulting from firewall misconfigurations due to an IPv4-to-IPv6 transitioning.

7.2 Incident Sharing Techniques

The following sections describe the realization of the mechanisms for the exchange of information between the FDH agents within a node, as well as between the FDH nodes in a particular network (area). Within the current thesis, the focus was set on plugging some basic techniques into the corresponding FDH components and using these techniques, in order to evaluate the overall feasibility of the FDH approach for distributed self-healing. Indeed, there is a vast amount of sophisticated research that deals with various techniques for information dissemination and routing over different network topologies, configurations and overlays [7]. All these techniques can be considered for the FDH dissemination components and used in concrete scenarios.

7.2.1 Introduction

The dissemination of alarms and incidents across the network (area) in question is the enabler for distributed Fault-Isolation and self-healing in the context of the FDH. The goal is to facilitate the FDH nodes to exchange information regarding their detected incidents and alarms, as well as regarding different events they have observed and experienced. That way the FDH nodes would

[6] As previously mentioned, further details on the technological implementation aspects - related to the FDH MonAgent and the plug-ins in this section - are provided in section 10.2

[7] Some of these works are cited in the following subsections.

strive to converge and compile a global picture with respect to problems within the network in question.

The alarm/incident/event sharing among the FDH entities is realized by a couple of FDH components. Given that an *FDH MonAgent plug-in* has an alarm/incident to pass to the other FDH components (also across the network), the belonging description is submitted to the *FDH repositories* of the local nodes, whilst in parallel it is directly passed to the belonging *FaultManAgent*, *SurvAgent* and *EvDissAgent* [8]. Thereby, the EvDissAgent is the node level component that serves as a gateway to the network, taking care of sending out the alarm/incident description, as well as responsible for receiving alarms/incidents from the network (i.e. alarms/incidents observed on remote nodes) and dispatching them to the local agents (i.e. FaultManAgent and SurvAgent) and to the local FDH node repositories.

A timely and efficient dissemination of alarms/incidents would enable the nodes implementing the proposed FDH self-healing framework to correlate faults/errors/failures/alarms beyond the scope of the local node, and to hunt down faults that propagate through the network. Therefore, diverse dissemination techniques need to be applied depending on the TCP/IP layer of the affected entities, on the severity of the alarms and incidents to be disseminated, and on the current situation in the network.

7.2.2 FDH Dissemination Techniques

In section 5.1.3 a number of requirements were identified which should be considered when selecting or designing the dissemination techniques for the FDH. Important aspects to consider within this section are given by:

1. the requirement for providing means for *alarm suppression* (**Req 13** from section 5.1.3 of the requirements chapter 5) similarly as in traditional network management, i.e.the alarm/incident/information should not be sent out to all FDH nodes across the network, but only to those which can utilize it and potentially identify root causes (faults) that can be taken care of by the FDH components residing on these nodes.

2. the requirement to incorporate multicast mechanisms and be able to deliver information to pre-defined groups of nodes (**Req 14**).

3. the need to select the appropriate dissemination strategy according to the urgency of the information to spread and the load in the network and the belonging nodes (**Req 15**).

4. naturally [9], unreliable communication protocols (**Req 16**), e.g. IP unicast/multicast or UDP, would be employed, which would mean that it is required to equip the dissemination mechanisms with means to increase the chances for successful delivery of the alarm/incident/event information despite the unreliable nature of the protocols.

In [CTS08], the dissemination of alarms and incidents was already discussed in the context of the UniFAFF [Cha09] framework, which was one of the starting points for the research presented in this thesis, along with the Omega architecture [BDmB06] and the research activities in the EU-FP7 EFIPSANS project [efi16b] [efi16a]. These techniques were implemented for the Autonomic Network Architecture (ANA) project [ANA16] [BJT+10], where the UniFAFF framework provided a

[8] All the agents reside on the local node, where the alarm/incident has occurred.

[9] The aspect of utilizing unreliable communication protocols for FDH is discussed in the following subsections.

platform for Fault-Management and Failure-Detection in the scope of the pursued clean slate approach. The ANA project aimed at redesigning the traditional ISO/OSI and TCP/IP stacks towards the provisioning of a network architecture with high degree of flexibility (i.e. through *Functional Composition*) and in-built management functionality. Thereby, the dissemination techniques were implemented on top of the link layer (in the form of Ethernet) ANA "compartment". A "compartment", which is an ANA concept, wraps different sets of entities (across the network) providing network functionality as a whole communication layer. These techniques from the ANA project were adopted for the IP based EvDissAgent of the FDH framework for distributed self-healing.

Three techniques have been adopted and tested for the FDH prototype. These are:

1. repetitive retransmission of the alarm/incident/event message by the EvDissAgent of the fault-detecting node

2. a canonical flooding on top of IP as well as a

3. gossiping on top of IP, similar as the one described in [CTS08] and [Tch09].

All these techniques are implemented based on datagrams, i.e. using unreliable communications protocols (e.g. UDP in case of the prototype of this thesis - see section 10.6). The usage of connection-oriented communication (e.g. TCP or the reliable Pragmatic Generic Multicast - PGM [LFE+01]) is a challenge in that context, since incidents/alarms/events can occur sporadically and the maintenance of a data stream is not so trivial, e.g. in PGM the delivery of a non-ACK [10] would not work reliably given the sporadic nature of the incident data to send. Next, each of the above listed dissemination techniques is elaborated on in turn.

Repetitive Retransmission: With respect to the first technique (the repetitive retransmission) from the above list, the fault detecting node is simply repetitively sending out messages (packed in datagrams) containing the alarm/incident/event description. These messages are posted to all the IP addresses (including uni-cast addresses as well as multicast groups - e.g. as in the scope of IPv6 based MLD [CV04] or IPv4 based IGMP [CDF+02]) which were pre-configured in the EvDissAgents accross the network. Thereby, the number of retransmissions as well as the delay between two consecutive retransmissions [11] must also be provided in advance, in case of employing this dissemination strategy. Correspondingly, this straight forward dissemination scheme can be summarized as follows.

[10] Within PGM, the participants in a multicast group must detect that they must have missed a packet in a sequence - which can be recognized based on the sequence numbers of the arriving packets - and notify the sender by generating a so called *non-ACK* packet.

[11] In the course of the experiments presented in this thesis, a number of 5 retransmissions and a delay of 0,5 seconds was used.

Dissemination Scheme (D1): Repetitive Retransmission of Alarm/Incident/Event Messages
Input: 1) $numberOfRetransmissions$ - the number of times each message should be retransmitted by the fault-detecting node. 2) $delayBetweenRetransmissions$ - the delay in seconds between two consecutive retransmissions. 3) $messageToDisseminate$ - the message to disseminate. 4) $IPAddressesForDissemination$ - a list of IP addresses to which the incident message must be sent out.
1. **foreach** i **in** $numberOfRetransmissions$ 2. **foreach** IP **in** $IPAddressesForDissemination$ 3. **send datagram** with $messageToDisseminate$ to IP 4. **end foreach**; 5. **sleep** for $delayBetweenRetransmissions$ seconds 6. **end foreach**;

Canonical Flooding on Top of IP: With respect to the flooding on top of IP, the prototype developed in the scope of the current thesis employes the dissemination scheme that was drafted in [CTS08] and [Tch09] [Tch10]. Thereby, the fault-detecting node sends out the alarm/incident/event message to all IP addresses which were pre-configured. The *EvDissAgents* on the nodes behind these IP addresses accept the message and check whether the contained alarm/incident/event description is already known locally, i.e. if it is already present in the local alarm/event/incident repositories or not. In case the alarm/incident/event description in the message is not known by the receiving node, it is stored locally and subsequently sent by the receiving/ forwarding node to all IPs which have been pre-configured in the *EvDissAgents* of the FDH nodes across the network. With respect to the dissemination node, the flooding over IP scheme can be summarized as follows.

Dissemination Scheme (D2): Activities of the Fault-Detecting Node for Alarm/Incident/Event Flooding on top of IP
Input: 1) $messageToDisseminate$ - the message to disseminate. 2) $IPAddressesForDissemination$ - a list of IP addresses to which the incident message must be sent out.
1. **foreach** IP **in** $IPAddressesForDissemination$ 2. **send datagram** with $messageToDisseminate$ to IP 3. **end foreach**;

Regarding the receiving/forwarding FDH node, the flooding over IP dissemination consists of the following steps:

Dissemination Scheme (D2): Activities of the Receiving/Forwarding Nodes in the Scope of Alarm/Incident/Event Flooding on top of IP
Input: 1) *IPAddressesForDissemination* - a list of IP addresses to which the incident message must be sent out.
1. **receive message from the network** 2. **if** *contentOfMessageNOTknownLocally* 3. **store content** in the local FDH repositories 3. **foreach** *IP* **in** *IPAddressesForDissemination* 3. **send datagram** with *contentOfTheReceivedMessage* to *IP* 4. **end foreach**; 5. **end if;**

Gossiping on Top of IP: Finally, a further dissemination scheme was used for delivering event messages (including alarms and incidents) to the FDH nodes across the network scope in question. This scheme was employed in cases when the events/alarms/incidents were not marked as urgent and hence they did not require a fast delivery across the network (e.g. LAN or OSPF area). For such cases, a gossiping over IP scheme was put in place, which was aligned to mechanisms and concepts presented in previous research works such as [Tch09], [Tch10], and [CTS08]. In these publications, this gossiping scheme was employed for a clean slate (i.e. non-IP legacy) type of approach to advancing network technologies. The gossiping scheme was implemented on top of the Ethernet compartment in an ANA (Autonomic Network Architecture) type of network.

The gossiping scheme is again to be described based on the actions to be undertaken by the initial sender (i.e. the FDH fault-detecting node) of the alarm/event/incident message, as well as by the actions required by the receiving FDH nodes. Following scheme describes the behavior of the FDH fault-detecting node that initiates the dissemination:

Dissemination Scheme (D3): Activities of the Fault-Detecting Node for Alarm/Incident/Event Gossiping on top of IP
Input: 1) *messageToDisseminate* - the message to disseminate. 2) *IPAddressesForDissemination* - a list of IP addresses denoting potential receivers. 3) *nIP* - the number of IP addresses contained in the *IPAddressesForDissemination* list. 4) *nIPstep* - the number of IP addresses to which the message should be sent out in each step.
1. *numberOfRounds* := *nIP* / *nIPstep* + *nIP* mod [12] *nIPstep* 1. **for** *i*:=0 **to** *numberOfRounds* 2. *IPaddr* := **pick** the next *nIPstep* addresses from *IPAddressesForDissemination* 3. **foreach** *IP* **in** *IPaddr* 4. **send datagram** with *messageToDisseminate* to *IP* 5. **end foreach**; 6. **end for**;

Regarding the receiving/forwarding FDH node, the gossiping over IP dissemination consists of the following steps.

[12] *mod* stands for a division resulting in the reminder of the normal division of two integer numbers.

Dissemination Scheme (D3): Activities of the Receiving/Forwarding Nodes in the Scope of Alarm/Incident/Event Gossiping on top of IP
Input: 1) $messageToDisseminate$ - the message to disseminate. 2) $IPAddressesForDissemination$ - a list of IP addresses denoting potential receivers. 3) $nIPstep$ - the number of IP addresses to which the message should be sent out if required.
1. **receive message from the network** 2. **if** $contentOfMessageNOTknownLocally$ 3. **store content** in the local FDH repositories 3. **for** i:=0 **to** $nIPstep$ 3. IP := **pick** the next address from $IPAddressesForDissemination$ 4. **send datagram** with $contentOfTheReceivedMessage$ to IP 4. **end foreach**; 5. **end if;**

The difference between the gossiping and the flooding scheme is given by the fact that the gossiping is less aggressive and sends out less datagrams to the network. The latter is due to the fact that the gossiping does not send the alarm/incident/event message to all IP addresses in the list, but only selects a subset of the pre-configured IP addresses. Hence, the gossiping can be used in cases where the network should not be demanded to an extreme, such as the dissemination of non-critical/-urgent alarms/events/incidents. In the current prototype of FDH, exactly the dissemination of non-critical/-urgent information is handled over gossiping, whilst critical messages are conveyed based on flooding. In addition, the repetitive transmission is also available as a dissemination scheme [13], and can be configured for all types of messages.

7.2.3 Evaluation of the Dissemination Quality

As described above, various dissemination schemes were implemented for the FDH proof of concept that was worked out in this thesis. This subsection focuses on evaluating and comparing the dissemination quality of two of the above presented mechanisms - the flooding over IP and the repetitive retransmission of datagrams. The gossiping was left aside, since it can be seen as a special version of the flooding for situations when the network is overloaded or for non-critical alarm/incident/event information.

The FDH canonical type of flooding on top of IP is described in the two algorithmic descriptions - marked as **Dissemination Scheme (D2)** - in the previous subsection. Thereby, whenever required, the symptom detecting node sends an alarm/incident/event message to a list of IP addresses (including multicast addresses as obtained within IPv6- MLD or IPv4-IGMP). In turn, each of the recipients forwards the message to all IP addresses in case it experiences the message for the first time, and drops it in case the message is locally known. This procedure worked sufficiently during the trials conducted in the testbed for evaluating FDH, which is described in chapter 11.

In order to evaluate the alarm/incident/event dissemination with respect to various properties on a large scale, an OMNET++ simulation was developed. In this subsection, the OMNET++ simulation results regarding the dissemination quality are taken a closer look at. Thereby, dissemination quality means the percentage of FDH nodes which are successfully informed/updated with respect to alarm/incident/event occurrences in larger networks, whilst considerable levels of background traffic are present in parallel. In addition, in section 12.2.2 in chapter 12, the same simulation setup

[13] Actually, this solution was implemented as first mechanism that was plugged into the *EvDissAgent* of FDH

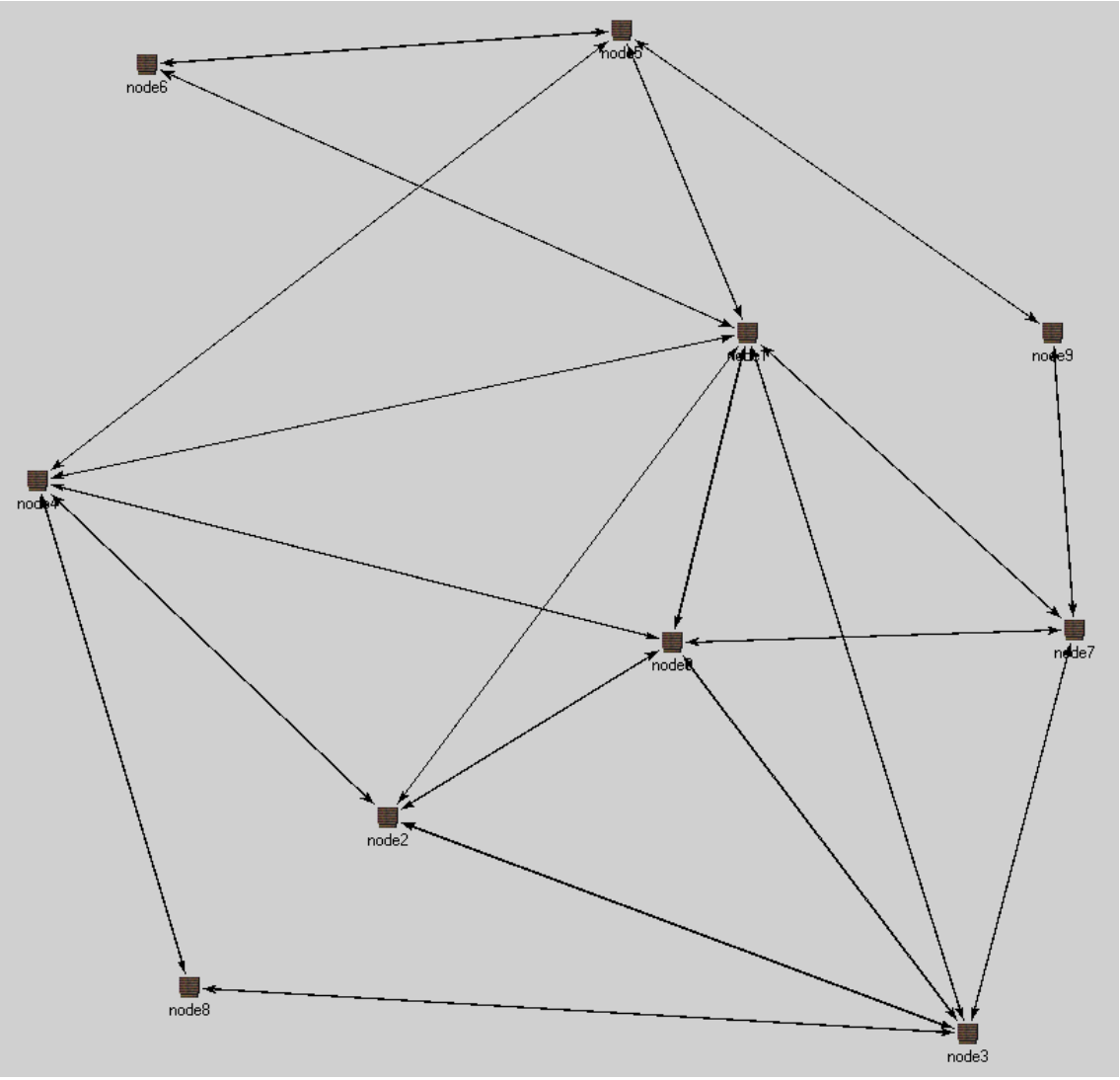

Figure 7.6: The 10 Nodes Network used for the Evaluation of the Incident Sharing Techniques as generated by the Waxman Model [Nal05] for Internet Topologies

is used for evaluating the scalability properties of flooding over IP and the repetitive transmissions scheme in large scale networks with deployed FDH instances in the network nodes.

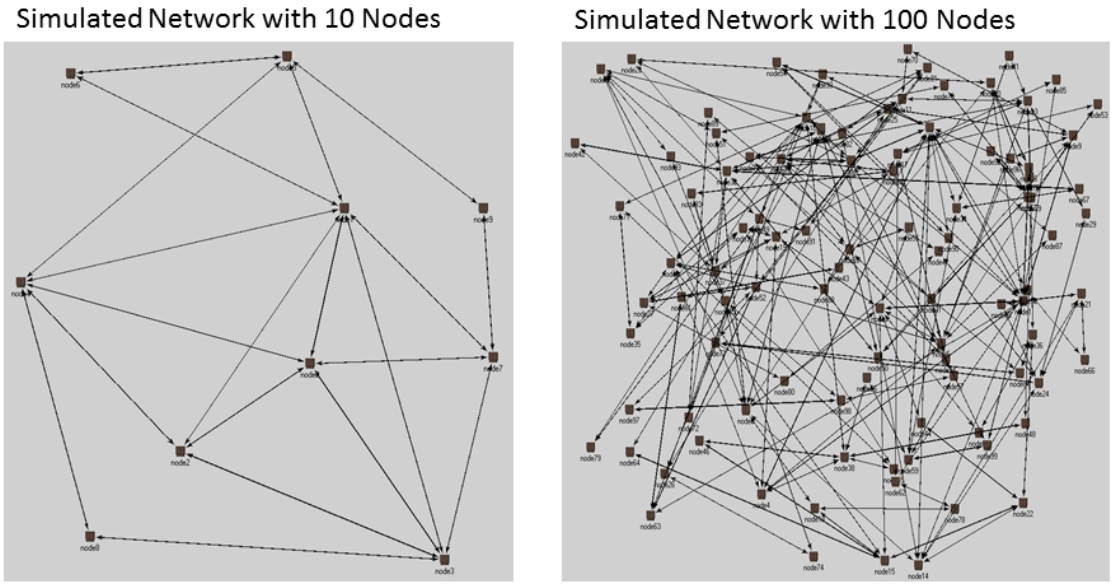

Figure 7.7: Networks used for the Evaluation of the Incident Sharing Techniques as generated by the Waxman Model [Nal05] for Internet Topologies

For the simulation setup, networks of various sizes were generated (10,100, 200, . . . ,1000) using the BRITE tool [BRI16] [MLMB01], which can employ different theoretical models, in order to output large scale Internet type of topologies. For the current, simulation setup the Waxman model [Nal05] was used, which resulted in a series of networks some of which are visualized in Figure 7.6 (for ten nodes in order to give a demonstrative example) and in Figure 7.7.

The conducted experiments compared the implemented flooding procedure with the repetitive retransmission of datagram incident messages by the fault detecting node. In addition, for the duration of an experiment, each node was pushing background traffic into the network thereby randomly selecting a node and sending a packet of size 1024 bytes every 100 microseconds. This resulted in at least a dozen of Giga-Bytes of background traffic for each simulated network. In that setup, there are two parameters of interest: the first one being the ratio of informed nodes - it should be noted that the case was simulated where no multicast addresses were used i.e. one IP address stands for exactly one router, and the second being the convergence time required to convey the incident message to all informed nodes. The latter aspect is elucidated on in section 12.2.2, where the scalability and overhead analysis of the various FDH components is presented.

The obtained results with respect to the ration of informed nodes are presented in Figure 7.8.Thereby, one can observe that the evaluated mechanisms scale reasonably up to 500 nodes when it comes to the percentage of informed nodes. For networks of larger sizes, the conducted experiments indicate that not all FDH nodes would be updated and able to converge with respect to alarms/incidents/events provided the conditions which were present during the experiments, i.e. amount of background traffic.

Certainly, during the past decades a large amount of research has been conducted on the area of gossiping and flooding. This research encompasses flooding and gossiping over special graph structures, e.g. Harary Graphs [LMM99], message routing in wireless sensor networks [BE02]

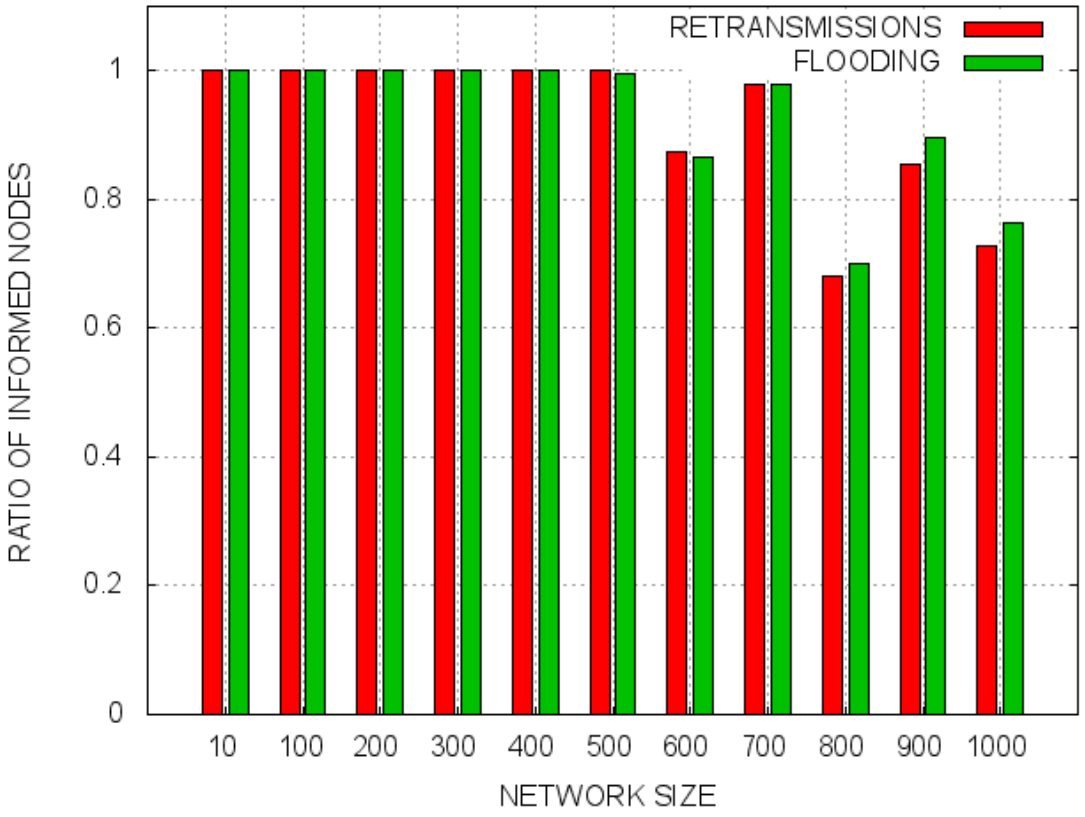

Figure 7.8: Ratio of Informed Nodes achieved during the Simulation of the Dissemination Techniques in Networks of various Sites

[Mar04], flooding and gossiping routing in MANETS [FGR+07] [HHL06] etc. Thus, the topic of alarm/incident/event dissemination has a great potential to grow (e.g. application of various research results) that is currently beyond the scope of the research in this thesis.

Finally, it should be noted that the techniques, which were used for the FDH *EvDissAgent* at the current stage, belong to the type of *in-band dissemination* schemes, i.e. they use the same network which carries the user/production traffic. It is also possible to use a dedicated network (e.g. some management or control network) for conveying the alarm/incident/event messages between the FDH instances. This type of dissemination/signaling is denoted as *out-of-band dissemination* and constitutes an additional option for improvement of the *EvDissAgent* quality and speed. However, the improvement achieved by using *out-of-band dissemination* comes at the cost of additional investment in infrastructure, i.e. dedicated signaling/control/management network.

7.3 Fault-Isolation

Next, the presentation of the key processes required for the self-healing function continues with addressing the process of Fault-Diagnosis/Localization/Isolation. Initially, a general introduction discusses on basic aspects of the Fault-Isolation task elucidated here. Subsequently, a scalable Markov Chain based algorithm for Fault-Isolation is presented[14], which was especially developed for the needs of the FDH framework, and proves to be scalable thereby determining key properties of the overall FDH based distributed self-healing. This algorithm is correspondingly compared to typically used Fault-Isolation schemes and discussed in the light of the FDH framework.

The coming introduction section elaborates extensively on the problem of Fault-Isolation and presents a number of approaches described in literature. Additionally, it characterizes the requirements for applying Fault-Isolation techniques in the context of FDH and prepares the reader for the description of the Fault-Isolation algorithms, to be deployed within the FIF functions.

7.3.1 Introduction

The term Fault-Isolation refers to the process by which the observed symptoms - alarms and failures - of an erroneous state are correlated, and a fault - root cause for the observed malfunctioning - is identified. Other names for this process that can be found in literature are Fault-Localization, Alarm Correlation, Event Correlation, and Root Cause Analysis [SS04a], as well as Fault-Diagnosis.

The process of Fault-Isolation is considered as one of the main challenges whilst executing Fault-Management processes[15] realizing network automation such as proposed in the scope of Autonomic Communications/Computing [KC03] and correspondingly Autonomic Fault-Management, as well as in the current scope towards the realization of a framework for distributed self-healing. The challenging character arises due to issues ranging from the complexity of the required models and the need for appropriate algorithms running on the models, to the performance characteristics of the whole machinery - *"Fault localization is subject to complications resulting from complexity, unreliability, and non-determinism of communication systems."* [SS04a]. Especially in the scope of FDH, the fact that *Root Cause Analysis* is considered as part of a distributed control loop, which

[14] The work presented here was initially published and presented at the IEEE Globecom conference [TCC10].

[15] For instance as defined by the FCAPS TMN standard [ITU00] or in the scope of ISO20000 based ITIL [ISO11]

is executed inside the network nodes, imposes special types of requirements regarding scalability such as computational performance and memory consumption of the process.

Fault-Isolation is considered, according to TMN [ITU00], as one of the main functions regarding Fault-Management. For that reason, intensive research has been conducted over the past years that investigated the application of a number of techniques from the field of Artificial Intelligence (AI), graph-theory and from the domain of modeling. Early initiatives such as [HSV99] aimed at defining a generic framework for Event Correlation. Bayesian networks [Bis06] were used for Fault-Isolation in 3G networks in [BDmB06] [BMQS08]. The usage of bipartite graphs (graph theory approaches) has been investigated in [KYGS07]. Furthermore, [SS04a] reviews approaches that have been applied or developed for the purpose of Event Correlation in telecommunication networks. A recent patent [BCF+17] provides an integrated framework for monitoring and correlating operational parameters towards pinning down sources of failures in computer networks and architectures. In other domains (e.g. Nuclear Power Plants and industrial manufacturing processes), the problem of Root Cause Analysis has been also addressed by the application of Hidden Markov Models (HMM) as investigated in [XG04] and [KKS02]. The application of Bayesian Networks for diagnosis in the aircraft domain was presented in [MBKM17] by discussing on how to obtain a reasonable Bayesian Network instance for the purpose of efficient analysis. Thereby, a mining approach is combined with human expertise in order to achieve best performance. These publications/works reflect just a fragment of the amount of conducted scientific activities related to the topic of Fault-Isolation. However, it was identified that this work normally targets either a specific problem (e.g. link failures), or provides methods which are not followed by an automated removal of the isolated fault, i.e. the Fault-Isolation is performed offline and the information gained is used to remove the fault manually. Because of the former fact, diverse real-time self-healing related aspects of the proposed methodologies were not really considered. Additionally, the mechanisms were not examined in the context of Resilience and distributed self-healing as proposed by FDH, targeting a generic approach to the diversity of potential network problems.

For the above reasons, as part of the current thesis, a Markov Chain based reasoning framework was developed, which is suitable for (close to) real-time purposes. A comparison between the proposed framework and the traditional widely used concept of Bayesian Networks is provided within the next paragraphs of this section. The understanding of these techniques requires basic knowledge from the area of linear algebra and probability theory on the level provided in [Bis06] and [DHS00].

The concept of Bayesian Networks (BN) is one of the main frameworks developed by the AI[16] community for the purpose of representing and traversing uncertain knowledge with a structure of causalities among the items of interest. Bayesian Networks constitute the mathematical foundation behind approaches to *Root Cause Analysis* such as *Causality Graphs* [SS04a] and *Dependency Graph* [SS04a]. A Bayesian Network, also called Belief Network, is a probabilistic graphical model expressed in terms of a direct acyclic graph (DAG). In such a graph, nodes represent random variables over a multivalued domain, and the edges indicate the dependencies between variables where the conditional distributions express the strength of these dependencies [SS04a].

The mathematical foundations of the Bayesian Network framework and related modeling approaches for Fault-Isolation are described in detail in chapter D within the appendix. The resulting inference problem is known to be NP-hard. Hence, it is obvious that Bayesian Networks bear a number of challenges when it comes to using them for fast and efficient distributed self-healing - especially with respect to performance, overhead and scalability. For that reason, an optimized

[16] Artificial Intelligence

algorithm is required that can be used for (almost) real-time reasoning regarding the root cause(s) of faulty conditions. Thus, the following subsections derive such a scalable Markov Chain based algorithm, which is also one of the key contributions of this thesis.

7.3.2 Scalable Markov Chain based Algorithm for Fault-Isolation

In order to systematically specify the proposed Fault-Isolation algorithm, the basic definition of a discrete Markov Chain as provided by [Ber93] is introduced:

Definition 7.3.2.1 *A discrete homogenous Markov Chain is a countable sequence of random variables* $(X_n)_{n\in N}$ *with values in a countable space* E*. The distribution of* $(X_1, X_2, X_3, \ldots)$ *has the following property:*

$$P(X_{n+1} = e_{n+1} \mid X_n = e_n, \ldots X_0 = e_0) = P(X_{n+1} = e_{n+1} \mid X_n = e_n) \tag{7.1}$$

with $e_0, \ldots, e_{n+1} \in E$.

(7.1) is denoted as (first-order [17]) Markov Property and $P(X_{n+1} = e_{n+1} \mid X_n = e_n)$ as the transition probability from e_n to e_{n+1}. The transition probabilities of a Markov Chain are captured in a stochastic matrix $P = (p_{ij})_{i,j\in E}$, in which the values in each row sum up to 1.

In Fault-Isolation, the aim is set at finding the most probable fault that is expected to have been activated in order to produce the monitored incidents, alarms and/or events in general. Therefore, the Markov Chain framework is looked at as a means for modeling event sequences that could have potentially caused an observed event. In this line of thought, set

$$E := \{\text{all known faults/errors/failures/alarms/events in the modeled network}\} \text{ with } \mid E \mid < \infty \tag{7.2}$$

and

$$F := \{\text{all known faults (root causes) in the modeled network}\} \text{ with } F \subseteq E. \tag{7.3}$$

In order to apply Markov Chains for the purpose of Fault-Isolation, the causality relationship is modeled that a particular error/failure/alarm has been caused by another incident/event with a certain probability. Thereby, the probabilities play an important role in case that an incident/event has been potentially caused by more than one event. In such cases, the probabilities give a ranking of the possible events directly causing the incident/event in question.

In the above setup, the value of the first random variable (X_0) in a sequence represents an observed symptom, i.e. $P(X_0 = \text{“observed symptom”}) = 1$. The most probable cause is diagnosed, when the probability distribution of a random variable in the sequence is concentrated over F and the probabilities are not changing for subsequent random variables, i.e. an invariant distribution has been reached. This is formalized as follows:

$$\forall\, i \in E \backslash F, \forall\, j \in F, \forall\, k \in E,$$

[17] First-order means that the probability for the values of a random variable are conditionally dependent only on the values of its predecessor random variable

$$P(X_n = i) = 0 \wedge \sum_{f \in F} P(X_n = f) = 1 \wedge P(X_n = k) = P(X_{n+1} = k) \tag{7.4}$$

If (7.4) holds, the most probable fault f^* is given by:

$$f^* = \underset{f \in F}{\operatorname{argmax}}\, P(X_n = f) \tag{7.5}$$

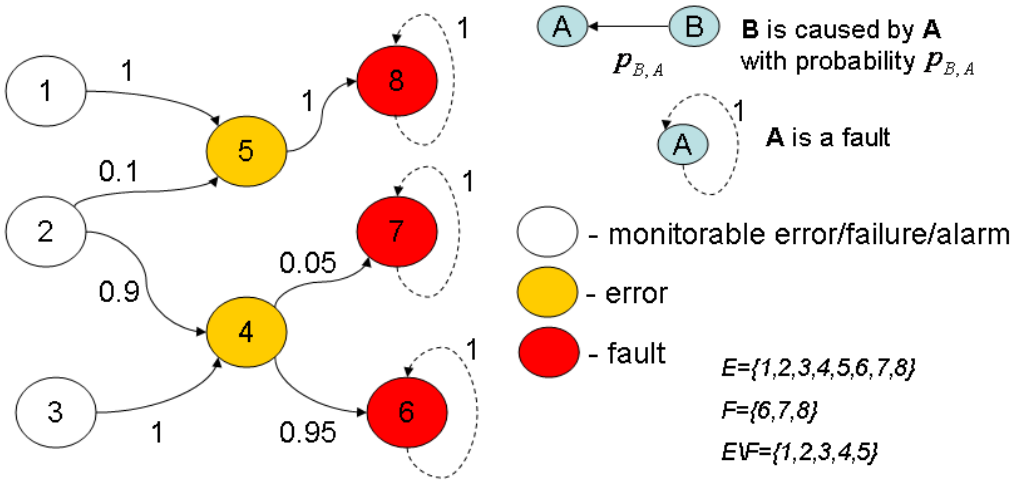

Figure 7.9: An example of a Markov Chain for Fault-Isolation

Naturally, rules are required regarding how faults/errors/failures/alarms must be embedded inside the Markov Chain framework, such that inference is facilitated, i.e. invariant distributions concentrating on F always exist. Hence, the following *Requirements* are derived:

Req. 1 (Modeling of Faults):
Faults (root causes for erroneous states) should be modeled as absorbing nodes

An absorbing node (events 6, 7 and 8 in Figure 7.9) in the Markov Chain representing graph is considered as a node having only one arc starting from it and ending at the same node. This edge has an assigned probability of 1.

Req. 2 (Modeling of Errors/Failures/Alarms):
The events from $E \backslash F$ must build up a DAG (directed acyclic graph), which means that no cycles are allowed in $E \backslash F$ and especially that none of the nodes representing events from $E \backslash F$ has an arc starting on the node and ending on the same node.

Req. 3 (Relations between Monitorable Events and Root Causes):
A path must exist from each element of $E \backslash F$ to at least one element of F.

Figure 7.9 gives an example of a Markov Chain that follows the requirements mentioned above. These requirements imply the convergence of the Markov Chain to an invariant distribution concentrating on F. This can be formulated as the following lemma:

Lemma: (Inference in Markov Chains for Fault-Isolation)
Given a discrete homogenous Markov Chain $(X_n)_{n\in N}$ with a stochastic matrix $P = (p_{ij})_{i,j\in E}$ that fulfils Req. 1-3, and a starting vector $\mu_{i\in E} = \delta_i$,then the Markov Chain converges to an invariant distribution that is concentrated on F. That is:

$$\exists\, n \in N, \mu_i\, P^n = \mu_i\, P^{n+1}$$

To illustrate the inference procedure, the Markov Chain in Figure 7.9 is represented as a stochastic matrix

$$P := \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.9 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.95 & 0.05 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{7.6}$$

Assume that incident 2 (Figure 7.9) has been monitored. Thus, $P(X_0 = 2) = 1$ and the reasoning starts with a distribution vector $\mu_2 = (0, 1, 0, 0, 0, 0, 0, 0)$. The theoretical foundations of Markov Chains allow to iterate over the sequence of random variables while iteratively multiplying the stochastic matrix P in addition to the starting distribution vector μ_2. That way, one can search for a invariant distribution implied by μ_2. Hence, the following limit is iteratively calculated:

$$\lim_{n\to\infty} \mu_2 \times P^n \tag{7.7}$$

For $n \geq 2$, it holds : $\mu_2\, P^n = \mu_2\, P^{n+1} = (0, 0, 0, 0, 0, 0.855, 0.045, 0.1)$, and according to (7.5): $f^* = 6$. Instead of going every time through matrix multiplications, the most probable faults (when starting in different events from E) can be pre-computed, stored in an array, and immediately retrieved during Fault-Isolation. That way, the proposed Fault-Isolation algorithm can be significantly optimized in terms of speed and memory consumption. However, the mere pre-computation presented here is only applicable in case a single symptom has been observed. In order to extend the algorithm for dealing with and correlating multiple observed events (e.g. failures, alarms, and events in general) the next section proposes a technique for exploiting the graph structure, which reflects the Markov Chain based Fault-Isolation model in question.

7.3.2.1 Extending the Markov Chain based Technique to deal with Multiple Symptoms

In this subsection, the Markov Chain framework for Fault-Isolation is extended, and the steps are defined that are required in the general case of $m \in N^+$ "simultaneously" reported observed incidents/events. In order to correlate the monitored events, the graph representation of the corresponding Markov Chain is exploited. First, it is required to search for the closest common predecessor of the reported symptoms. The term "closest common predecessor" stands for the event-representing node with the minimal distance (in terms of number of edges) to the observed incident events (symptoms). The closest common predecessor is a joint event that explains and represents a set of reported incident events. Once such a predecessor is found, the reasoning process can proceed as presented in the previous section, i.e. the most probable pre-computed root

cause should be selected with respect to starting the Markov Chain in the identified closest predecessor. In case the reported incidents do not have a common predecessor, they must be sorted in a number of sets of maximum size, each of which having its own common nearest predecessor.

Given that the shortest path matrix A of the Markov Chain representing graph is available (e.g. obtained by using [Flo62]), then the matrix A^T gives the distance between all pairs in the direction from the faults (root causes) to the symptoms. The positions of none zero entries in the rows of A^T which represent the set F (faults) in the graph, provide the errors/failures and alarms which are caused by this fault, e.g. in Figure 7.9 fault 8 causes error 5 and monitorable incidents 1 and 2. The events caused by a particular fault are intrinsically guaranteed to have a common predecessor. For the purpose of run-time Fault-Isolation, these $\mid F \mid$ (number of faults) sets of events (caused by each single fault) are stored in an appropriate structure and used during the on-line Root Cause Analysis. Hence, whenever a set I of $m \in N^+$ events (incidents) are "simultaneously" reported, the reported incidents must be grouped together that build up a subset of each of the pre-computed event sets originating from each single fault. A single reported incident could participate in multiple such groups. The sets of grouped reported events are denoted as $J := \{J_1, \ldots, J_{|F|}\}$. Every set of J is guaranteed to have a common predecessor. Thus, Fault-Isolation must be performed separately for each single element of J that is not empty. The next algorithm is used to construct J.

Algorithm (A1): Extract from the reported incidents the sets that have a common predecessor
Input: 1) T - a number of $\mid F \mid$ sets which contain the errors/failures/alarms that are caused by each single fault from F: $T = T_1, ..., T_{\mid}F \mid$. 2) I - the set of reported incidents with $I = \{i_1, ..., i_m\}$
Output: J - a set of groups of events from I such that each group is a subset of the sets from T with $J = \{J_1, \ldots, J_{\mid F \mid}\}$.

```
1. init J_1 = ∅, ..., J_|F| = ∅ , and J = J_1, ..., J_|F|
2. foreach i in I
3.         for k = 1 to | F |
4.                 if i ∈ T_k then
5.                         J_k = J_k ∪ {i};
6.                 end if;
7.         end for;
8. end foreach;
```

After A1 has identified the subsets of I that are guaranteed to have a common predecessor, another iteration is required, in order to find the belonging closest common predecessors. This is done by the following algorithm.

Algorithm (A2): Find the closest common predecessor of a set of incidents which are guaranteed to have a common predecessor

Input:
1) K - a number of reported incidents that have a common predecessor
2) A - a matrix containing the shortest paths between all pairs of nodes inside the graph. If there is no path between two nodes, then the corresponding matrix entry is 0.

Output: cP - the closest common predecessor for all reported incidents from K

```
1. init S = ∅
2. select randomly one k from K
3. for i = 1 to | K |
4.         if A_kj ! = 0 then
5.                 found = TRUE;
6.                 foreach j in K\{k}
7.                         if A_ji == 0 then
8.                                 found = FALSE;
9.                                 break;
10.                        end if;
11.                end foreach;
12.                if found == TRUE then
13.                        S = S ∪ {i};
14.                end if;
15.        end if;
16. end for;
17. cP = argmin_{j∈S} A_kj;
```

A2 picks one of the incidents from the set with the common predecessor, and searches for a predecessor - an event that must have happened before the incident according to the Markov Chain representing graph. Once such a predecessor is found, the algorithm checks whether it is also a predecessor for the rest of the reported incidents in K.If this is the case, then the predecessor is added to the set of common predecessors. Finally, the closest common predecessor is selected as the one that is the closest to the reported incident selected in the first lines of the algorithm. This is possible since due to the DAG structure of the sub-graph on $E\backslash F$, the closest common predecessor of one of the events in K is automatically the closest common predecessor of all elements of K. Finally, the reasoning proceeds as derived for a single reported event, starting with the common predecessor as described in the previous section.

Having specified the different pieces of the Markov Chain based Fault-Isolation technique, the next section proceeds with the theoretical time and space complexity analysis of the proposed algorithm.

7.3.2.2 Complexity Analysis

The time complexity of A1 is *O(|F||I|)*. A2 has a time complexity of *O(|E||K| + |E|)*, because of the embedded cycle and the obtaining of cP at the end.In the worst case K can reach the size of *|I|*. Hence the time- complexity is given by *O(|E||I| + |E|)*. A2 must be run *|F|* times in the worst-case in order to find the closest common predecessor for each of the identified sets $J = \{J_1, \ldots, J_{|F|}\}$. Together with the complexity of A1 (A1 is required to run before A2 for the purpose of run-time

inference), a time complexity of $O(|F|(|E||I|+|E|)+|F||I| + 1)$ results as a first complete estimation. The "1" at the end results from the additional effort that is needed in order to select the most probable root cause from the corresponding pre-computed array. This estimation is dominated by $|F||E||I|$. Hence, this implies a worst-case time complexity of $\boldsymbol{O(|F||E||I|)}$ for run-time reasoning, considering that the fault/error/failure/alarm relations have been modeled within our proposed Markov Chain based framework.

Another concern regarding scalability is the amount of memory required by the proposed algorithm. From the above description of the algorithms, it is possible to identify the following data required for Fault-Isolation:

1. A vector of size $|E\backslash F|$ that stores the most probable root cause for each monitorable error/failure/alarm
2. A matrix with the shortest paths (inside the Markov Chain representing graph) between every two events - such a matrix is of size $|E|^2$
3. A number of $|F|$ sets of events originating from each single fault - in the worst case $O(|F||E\backslash F|)$

Hence, the overall required space complexity is $O(|F||E\backslash F|+|E|^2+|E\backslash F|)$. Taking $|E|$ as an upper bound for $|E\backslash F|$ results in a worst-case space complexity of $\boldsymbol{O(2|E|^2+|E|)}$.

To summarize: based on the analysis conducted in the course of this section, one can observe that the proposed Fault-Isolation scheme has a polynomial time and space complexity. This fact allows to expect fast Fault-Isolation within the FDH nodes with minimal overhead in terms of resource consumption (e.g. memory).

7.3.3 Example Application: Localizing Ethernet Duplex Mismatch and potential Black Holes in IPv6 networks resulting from IPv4-to-IPv6 Transitioning

The case study builds on the two network problems which were already used for the evaluation of the FDH MonAgent in section 7.1.3. In that sense, section 7.1.3 already describes quite extensively the specifics of the network problems in questions, whilst the following subsection focuses on the aspects, which are of relevance for evaluating the Fault-Isolation procedure. Further details on these selected networking problems are provided in chapter 13 for the purpose of evaluating the overall operation of the FDH self-healing framework.

Within the presented case study, the proposed algorithm is used to localize the problems of potential Black Holes [KYGS07] [Lah00] in IP networks (especially in the scope of IPv6) and Ethernet Duplex Mismatch [SC05]. Thereby, the effectiveness of the proposed Markov Chain based technique is compared against the commonly used Bayesian Network based concept of a Causality Graph [SS04a], which is similarly applied on causality relations among events - faults/errors/failures/alarms.

7.3.3.1 Problem Definition

As previously described: Black Holes, as elucidated on in [Lah00], can potentially occur in IPv6 networks in case some routers (Black Hole routers) have been wrongly provisioned with IPv4

relevant firewall configurations suppressing ICMP messages. Provided that the PMTU (Path Maximum Transmission Unit) of a flow decreases at the Black Hole router during the lifetime of a connection - e.g. because of a link failure and subsequent rerouting leading to a lower PMTU value, the Black Hole router would be subsequently unable to forward packets on the link with a smaller MTU (there is no packet fragmentation on routers in IPv6) and would not be able to send an ICMPv6 error notification to the sender (because of the aforementioned ICMP suppression in the Black Hole router firewall). The sender in turn would not able to readjust the size of the packets being sent out, which would lead to a timeout of the connection. Regarding the Ethernet Duplex Mismatch issues, the elucidations given in [SC05] are used, which describe the situation of two Fast- Ethernet Network Interface Cards (NIC) that fail to auto-negotiate the duplex mode on a link. The aim in the current context is to identify the correct fault occurrence (misconfigured firewall or wrong duplex mode of a NIC) based on different monitored incident events. A more elaborated discussion of the networking problems is given in chapter 13, where the case studies are described, based on which the overall FDH machinery is evaluated. Additionally, it should be noted that section 7.1.3 already provided a more elaborated description of the networking issues in question, however mainly from the perspective of the FDH monitoring framework.

7.3.3.2 Constructing a Fault-Propagation Model

Assuming that the network operation personnel administers a network as the one in Figure 7.5, the first thing that stands out is that this network is a compilation of different types of links. Hence, the network administrator would recognize the potential for a Black Hole at R2 and for a Duplex Mismatch on the links R2↔R3 and R3↔R6. The network operation personnel is expected to extract a number of potential incidents out of [SC05] and [Lah00] and construct a Fault-Propagation Model which describes the relations among these events. In practice, the process of constructing a Fault-Propagation Model would be realized by using the available knowledge about the causes of different types of network problems. This knowledge can be obtained from the community (e.g. publications, white papers, discussions ...), by performing different tests (such as conformance, stress and penetration testing of protocols), as well as from simulations and validations of protocol or system behaviors. The structure of the Fault-Propagation Model for the Fault-Isolation case study - encompassing 30 events - is illustrated in Figure 7.10.The events are listed in Table 7.1. The graph in Figure 7.10 is equivalent to the underlying directed acyclic graph of the Bayesian Network for this problem. The structure of the corresponding Markov Chain is obtained by reversing the arcs of the graph in Figure 7.10.

7.3.3.3 Experimental Setup

The proposed Markov Chain based Fault-Isolation algorithm was implemented in C++ and evaluated on a Linux machine (Intel(R) Core(TM), 2 x"2.80GHz Duo CPUs", 3.9 GB RAM). The Bayesian Network evaluations and comparisons are based on the open source Probabilistic Networks Library (PNL) [PNL16] which was initiated by Intel.

7.3.3.4 Training and Evaluation Sets

In order to evaluate the Fault-Isolation qualities of the proposed framework, training data as well as evaluation data sets have to be designed. The training data is used to obtain the probability based parameters of the underlying Markov Chain and/or Bayesian Network, whilst the distinct

Table 7.1: Events inside the constructed Fault-Propagation Model

Nr.	TYPE	DESCRIPTION
1	FAULT	The NIC on R2 towards R3 operates mistakenly in half- dulpex mode.
2	ERROR	R3 fails to send some frames to R2 because of interferences with the R2 NIC mistakenly operating in half-duplex mode (for more details see [SC05]).
3	ERROR	TCP packets flowing from R3 to R2 get lost.
4	ERROR	Duplicate ACKs in TCP flows streaming over R3 and R2 in a row.
5	ERROR	Increased number of duplicate ACKs in TCP connections flowing over R2 and R3.
6	ERROR	Increased number of dropped packets at the R3 NIC connecting R2 and R3.
7	FAULT	The NIC on R3 towards R2 operates mistakenly in half- dulpex mode.
8	ERROR	R2 fails to send some frames to R3 because of interferences with the R3 NIC mistakenly operating in half-duplex mode (for more details see [SC05]).
9	ERROR	TCP packets flowing from R2 to R3 get lost.
10	ERROR	Duplicate ACKs in TCP flows streaming over R2 and R3 in a row.
11	ERROR	Increased number of dropped packets at the R2 NIC connecting R3 and R2.
12	FAULT	Misconfigured firewall suppressing ICMP packets on R2
13	ERROR	R2 is unable to send ICMP messages.
14	ERROR	R2 fails to send ICMP Packet Too Big notification to the corresponding end system
15	ERROR	Incorrect PMTU assumed on the end system.
16	ERROR	Large TCP packets fail to pass R2.
17	ERROR	Duplicate ACKs in the direction from R1 into the network.
18	ERROR	Duplicate ACKs in the direction from the network to R1.
19	ERROR	Duplicate ACKs observed on R1.
20	FAULT	The NIC on R3 towards R6 operates mistakenly in half- dulpex mode.
21	ERROR	R6 fails to send some frames to R3 because of interferences with the R3 NIC mistakenly operating in half-duplex mode (for more details see [SC05]).
22	ERROR	Increased number of dropped packets at the R6 NIC connecting R3 and R6.
23	ERROR	TCP packets flowing from R6 to R3 get lost.
24	ERROR	Duplicate ACKs in TCP flows streaming over R6 and R3 in a row.
25	ERROR	Increased number of duplicate ACKs in TCP connections flowing over R3 and R6.
26	FAULT	The NIC on R6 towards R3 operates mistakenly in half- dulpex mode.
27	ERROR	R3 fails to send some frames to R6 because of interferences with the R6 NIC mistakenly operating in half-duplex mode (for more details see [SC05]).
28	ERROR	Duplicate ACKs in TCP flows streaming over R3 and R6 in a row.
29	ERROR	Duplicate ACKs in TCP flows streaming over R3 and R6 in a row.
30	ERROR	Increased number of dropped packets at the R3 NIC connecting R6 and R3.

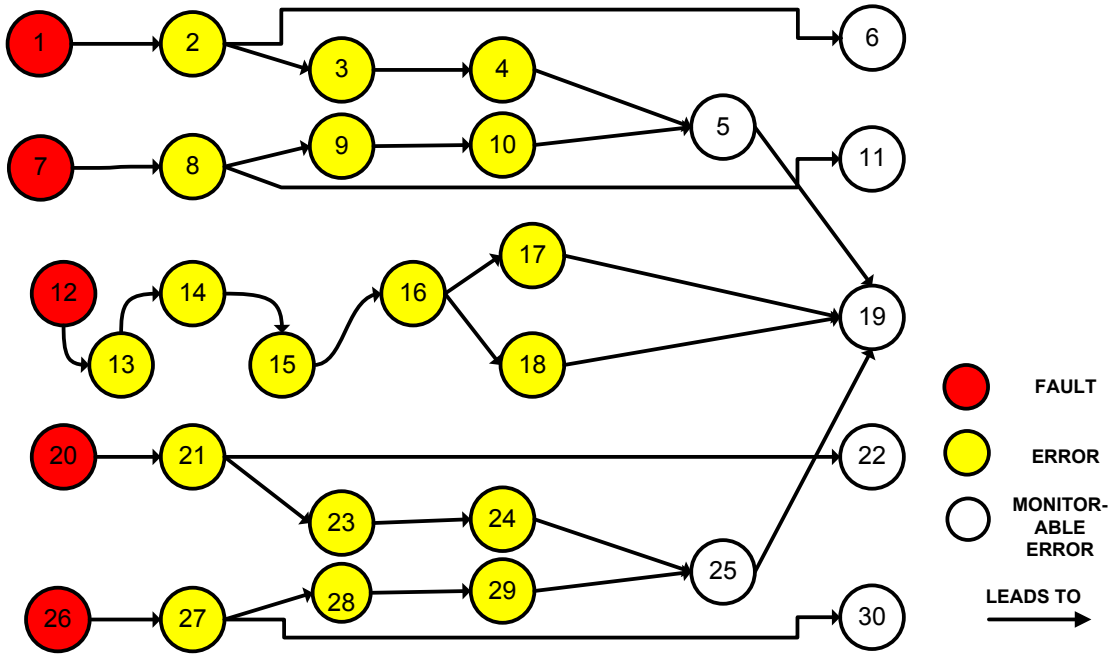

Figure 7.10: The structure of the constructed Fault-Propagation Model

evaluation/test data sets are needed, in order to check how good the reasoning frameworks can identify the root causes for faulty conditions within the current case study.

In order to design the training data, a number of 20000 faults were picked from the Fault-Propagation model according to the discrete uniform distribution over the set of faults F. The resulting chains (e.g. 1→2→3→4→5→19 and 1→2→6) of events according to the Fault-Propagation Model (Figure 7.10) were extracted and used for training the Markov Chain and Bayesian Network based models.

To create evaluation sets, 6000 multiple simultaneous fault activations were simulated which resulted in a total number of 17960 faults and belonging symptoms. For example, in case the simultaneous occurrence of 1 and 7 was simulated, the resulting monitored events are given by 5,6,11,19. These symptoms were then given to the Bayesian Network and to the Markov Chain based Fault-Isolation technique in order to evaluate their fault identification properties, which were trained based on the previously described training data. For example, if the event set $I = \{5, 6, 11, 19\}$ is submitted as monitored events to the Fault-Isolation frameworks, it is expected that in the optimal case the corresponding reasoning algorithms should identify the set $\{1, 7\}$ as simultaneously activated faults.

7.3.3.5 Obtaining Key Parameters of the Evaluation Models

Based on the above described training set, the corresponding Maximum Likelihood estimators were used to train the Markov Chain and respectively the Bayesian Network. For the Markov Chain framework the Maximum Likelihood Estimator is given by

$$\hat{P}_{ij} = \frac{n_{ij}}{n_i} \tag{7.8}$$

according to [Teo09],with $\hat{P}_{ij}$ being the estimated transition probability from event i to event j, n_i being the overall number of transitions (within the training event sequences) starting from event i, and finally n_{ij} being the number of transitions from event i to event j within the training set.

For training the Bayesian Network based Causality Graph, the Maximum Likelihood Estimator (MLE) [PNL03] [JB00] provided by the PNL library was used. This MLE is defined as follows

$$\hat{p}(x_v \mid x_{\pi_v}) = \frac{m(x_{\phi_v})}{m(x_{\pi_v})} \tag{7.9}$$

for an event/incident node v from the Bayesian Network constituting the base for a Causality Graph. Thereby, π_v stands for the set of direct predecessors of the node v according to the structure of the underlying Bayesian Network, x_v stands for a particular value of the node v, whilst x_{π_v} denotes a particular tuple of values for the predecessor nodes of v in the Bayesian Network. In the case of a Causality Graph for Fault-Isolation, an event representing node may only take the values 0 - meaning that the event has not occurred, and 1 - meaning that the event has occurred. Furthermore, $m(x_{\pi_v})$ stands for the total count of occurrences of the particular tuple of values x_{π_v} in the training set, whereas $m(x_{\phi_v})$, with $\phi_v = \pi_v \cup \{v\}$, denotes the number of occurrences of x_{π_v} in conjunction with the particular value x_v of the node in question (i.e. v). Hence, the estimation of the probability for the particular value x_v of node v, subject to a particular tuple of values x_{π_v} for its predecessors, is given by the ratio between the number of occurrence of x_v in conjunction with x_{π_v}, i.e. $m(x_{\phi_v})$, divided by the total number of occurrences of x_{π_v}, i.e. $m(x_{\pi_v})$.

Based on the above described Maximum Likelihood Estimators and the pre-designed training data set, the probability parameters of the Markov Chain and of the Bayesian Network for the Causality Graph were obtained. In the case of the Bayesian Network, the Maximum Likelihood Estimation functions of PNL [18] were used. The number of parameters that had to be learnt was 30x30=900 in case of the Markov Chain and 146 conditional probabilities in case of the Bayesian Network. The evaluation set was used to compare the performance of the proposed method with the Bayesian Network based Causality Graph.

7.3.3.6 Performance Measure

Before issuing a comparison between the qualification properties of the Fault-Isolation techniques, the KPIs should be clarified upon which the comparison takes place. A typical measure, which is used in the areas of statistics and machine learning, for evaluating the properties of a classifier is given by the *F-measure* [Rij79]. The F-measure is defined as follows:

$$F = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{7.10}$$

with $Precision$ being the ratio of correctly identified root causes (i.e. faults) from the overall set of faults, which were deemed as root causes for the observed events, i.e.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{7.11}$$

Moreover, the $Recall$ is defined as follows

[18] These are based on the above described technique.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{7.12}$$

thereby reflecting the ratio of correctly identified faults - i.e. root causes for the observed alarm/incident events - from the sum of the correctly identified root causes and the wrongly unidentified root causes - i.e. the cases in which a fault has occurred but was not identified as a root cause by the corresponding Fault-Isolation technique. The F-measure is a widely used as a performance indicator for evaluating classifiers and diagnostic techniques, since it combines the *Precision* and *Recall* in a harmonic type manner. Thereby, *Precision* and *Recall* are two key aspects with respect to which a classifier should be optimized.

Based on the above consideration on *Precision* and *Recall*, the $F - measure$ can be summarized as follows:

$$F = \frac{2 \times True\ Positives}{2 \times True\ Positives + False\ Positives + False\ Negatives} \tag{7.13}$$

7.3.3.7 Evaluation of Classification Properties

In the scope of the Case Study presented here, the Bayesian Network based Causality Graph achieved an F-measure value of 0,888786 while the proposed Markov Chain based technique achieved an F-measure value of 0.896162. This means that the proposed approach to Fault-Isolation performed slightly better in terms of fault identification quality. In addition, it can be expected that the Markov Chain based Fault-Isolation (proposed in this thesis) would have some more advanced scalability properties as compared to traditional Bayesian Networks. This is verified in the coming section.

7.3.3.8 Empirical Scalability Evaluation and Comparisons

Next, a scalability evaluation and comparison between the proposed Markov Chain based approach and the BN framework is presented, with respect to a growing number of nodes in the network - managed by FDH in regard to self- healing - and corresponding events in the Fault-Propagation Model. To simulate such growth, the network in Figure 7.5 was replicated by copying it and attaching the copy to the corresponding router at the edge next to the user's end system. In addition, the Fault-Propagation Model was changed according to the resulting network. Thus, the size and structure of the FPM was extending with respect to the number of events. The aspects that were investigated, and based on which a comparison was drawn, were given by the time needed to perform Fault-Isolation, as well as by the memory consumption, which was observed in the light of a growing number of events. The results can be seen in Figure 7.11 and Figure 7.12 and clearly indicate that the Markov Chain based approach outperforms the PNL implementation of Bayesian Networks with respect to time and memory scalability. The comparison between the Markov Chain approach and the Bayesian Network stops at a number of 165 events due to the inability of the PNL library to cope with larger Fault-Propagation Models. These issues were investigated with the valgrind [NS07] [VAL16] memory analyzer that indicated a number of memory allocation errors within the PNL library.

To show further that the Markov Chain based approach (proposed in this work) can be used in real time for large networks, additional experiments were conducted, the results of which are presented

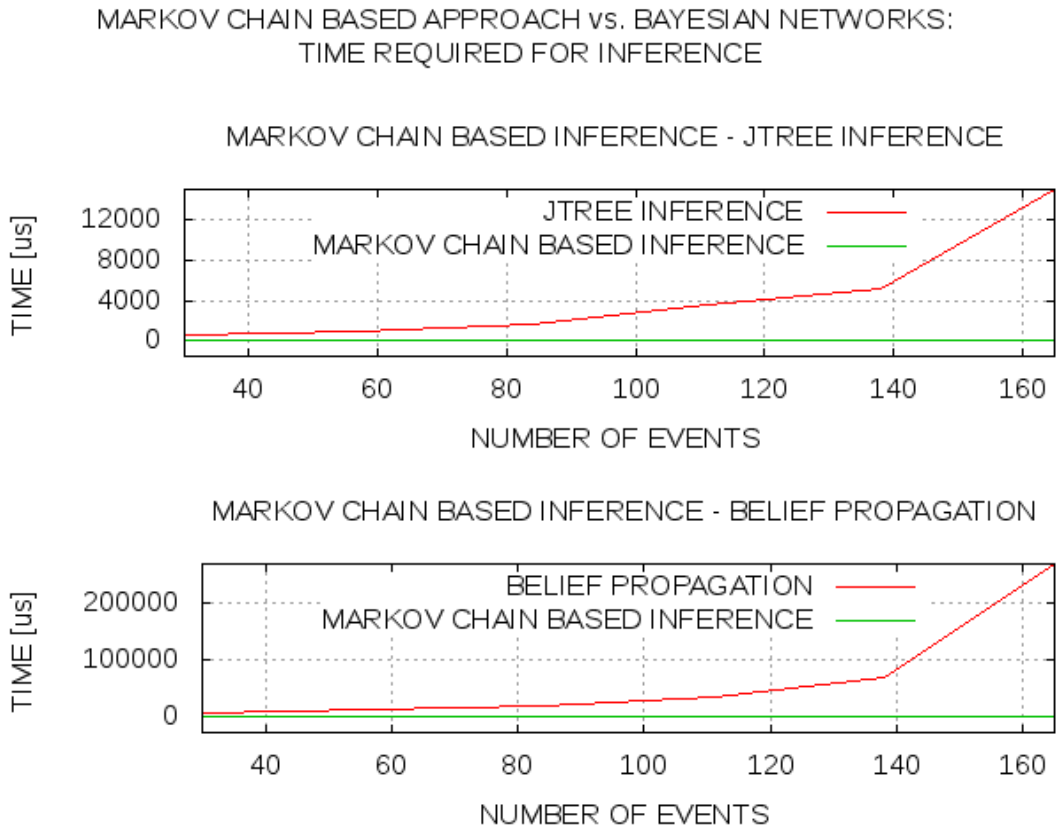

Figure 7.11: Markov Chain based approach vs. Bayesian Networks: Time required for Inference

and discussed in the dedicated chapter on FDH scalability (see section 12.2.3).

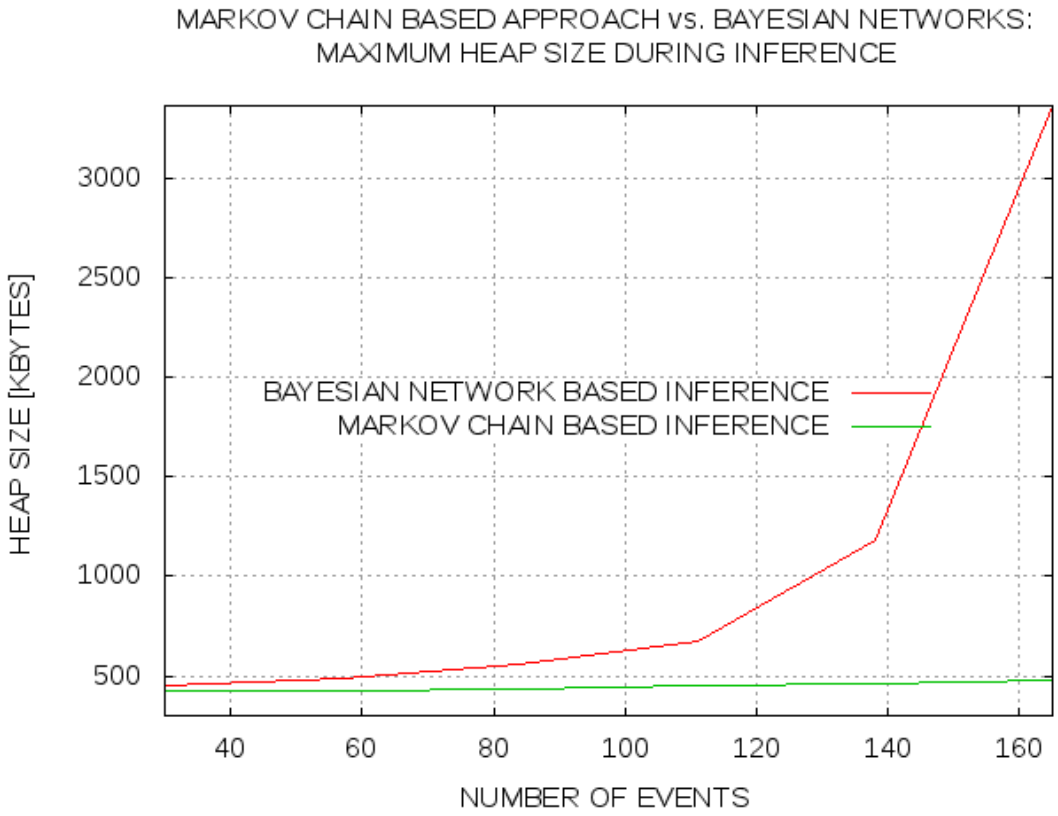

Figure 7.12: Markov Chain based approach vs. Bayesian Networks: Maximum Heap Size during Inference

7.3.4 Discussion

In the course of this thesis, a novel technique for Fault-Isolation was specified and evaluated [19]. This technique is based on the mathematical foundations of Markov Chains and is used for probability based reasoning and identification of root causes for faulty conditions. The resulting reasoning scheme was compared to the Bayesian Network based Causality Graph framework, which is a traditionally applied technique [SS04a] in networks' Fault-Management and serves similar purposes.

Looking on both frameworks from the theoretical point of view, the comparison can be based on the (time and space) complexities derived above, as well as on the properties of the corresponding model. As previously mentioned, a Bayesian Network is *NP-hard* in the general case. For special cases of Bayesian Networks, there exist efficient heuristic algorithms that can approximate an optimal solution of the Fault-Isolation problem. On the other hand, the Markov Chain framework has a better time complexity of $O(|F||E||I|)$ [20] in the general case. Hence, it can be claimed that in the latter aspect the Markov Chain approach is better, since: 1) it does not im-

[19] The current discussion is based on the findings initially presented at the IEEE Globecom conference [TCC10]

[20] Please refer to section 7.3.2 for the exact definition of the alarm/incident/event sub-sets F, E and I

pose any additional special requirements on the type of applied instances, and 2) it appears to be faster based on the derived computational complexity. Beyond these theoretical considerations, the argumentation with regard to speed could be confirmed by the measurements and numerical comparisons which were conducted and described in the course of section 7.3.3.8. Furthermore, with respect to space complexity, the Markov Chain based approach scales quadratically in the number of alarms/incidents/events whose relations are captured. In the Bayesian Networks case, a conditional probability matrix (in the "very"worst case already nearly quadratic in the number of alarms/incidents/events) should be stored for each single event, which implies a worst-case cubic space complexity - quadratic number of relations pro event times the overall number of events. For comparison, the proposed Markov Chain based approach comes with a quadratic time complexity, as derived in section 7.3.2.2. Hence, one would also expect the Markov Chain based Fault-Isolation to require less memory than the Bayesian Network based one. This assumption is clearly confirmed by the numerical results in section 7.3.3.8, which analyzes the heap memory consumption during the operation of both algorithms.

Despite the above described Bayesian Network drawbacks, it can be claimed that BNs have a big advantage over Markov Chains with respect to their modeling properties regarding the relations among alarms, incidents and events. In the case of the proposed Markov Chain based framework, the information about the occurrence of intermediate events in a fault- propagation chain is totally neglected. The proposed (Markov Chain related) algorithms are designed in a way that they address only the isolation of a particular fault without identifying sequences of events that could have probably originated as a result of this fault. Hence, this information would not be available if it is required within the process of Fault-Removal. On the contrary: the Bayesian Networks compute a probability of occurrence for every single event (alarm/incident). Thus, they provide the means to estimate the whole amount of alarm/incident events (root causes, intermediate events and symptoms) that must have occurred under the current faulty condition.

The preceding discussion clearly indicates that the BNs outperform the Markov Chain based framework proposed in this thesis, when it is required to reason not only for faults, but for the overall set of relevant events that must have occurred and constitute the current erroneous state. However, Bayesian Networks have a number of disadvantages with respect to their time complexity ($NP - hard$), the restrictions that must be imposed on their structure to enable scalable reasoning algorithms to run, as well as the space complexity of a Bayesian Network based frameworks, e.g. Causality Graphs.

7.4 Failure Prediction

The next aspect which requires elucidation is given by the task of Failure-Prediction which is to be implemented by the FPF functions of the SurvAgent.

7.4.1 General Considerations

As specified in section 6.2.1.2, the goal of the FPF is to acquire various monitoring information, including monitored events, to analyze those and anticipate the activation of faults and their manifestation as failures ahead in time. Thereby, the FPF is designed in a way as to be able to utilize different algorithmic frameworks and models for the task of Failure-Prediction. Indeed, the monitoring information that is analyzed is received from the FDH MonAgent within a network node, and depends on the type of utilized monitoring plug-ins. Therefore, the FPF module of the

SurvAgent should send the corresponding monitoring requests to the node level FDH MonAgent during the initialization phase, in order to receive the required information.

In case the requested information [21] consists of events, which can be analyzed similarly as in the case of Fault-Isolation, the application of the techniques which were discussed in section 7.3 can be considered. These include the presented Bayesian Network framework as well as the Markov Chain algorithm for fast and scalable reasoning. In that case the relations between events would need to be modeled, in terms of the possibility and likelihood for particular observed events, to be the indication of a fault activation and belonging propagation to (a) service affecting failure(s) in the near future. This is in contrast to the modeling which was presented in the section on Fault-Isolation (section 7.3, where the employed models were semantically capturing the likelihood for particular events to have been caused by other events.

7.4.2 Possible Techniques for Failure-Prediction

Given the above general considerations, it is possible to deploy various other frameworks (other than the Markov Chain or Bayesian Network based approaches so far) for the goals of Failure-Prediction. An example for such framework is given by the so-called Hidden Markov Models (HMM) [Rab90] [SMA+16], which are closely related to the Bayesian Network or the Markov Chain concepts. In the case of HMM, the relation between various observed events and the possible implications of these events for an internal/hidden model of the system is calculated. These implications to the internal /hidden model can be interpreted as expected to lead to anticipated fault activations and the belonging manifestation of failure events.

Additional techniques and frameworks from the areas of machine learning and mathematical statistics may be employed for the needs of the FPF. Such techniques are given by traditional classifiers such as Linear [YLTP16] [YS09] or Ridge Regression [vW17], Support Vector Machines [YJH+16], Gaussian Processes [BWA16] [RW05] or by the widely used Neural Networks [SRKO16] and Self-Organizing Maps [Koh90]. Thereby, these techniques would be operating on numerical values regarding different key performance indicators and would be trying to anticipate various potential anomalies based on their embedded model. This model is normally represented as parameters of the corresponding technique. The parameters would be obtained based on training sets characterizing the typical operation of the network/system.

Another interesting proactive approach to router failures is presented in [VNC12] [VCNR13] and [WTVL13]. Thereby, based on node failure characteristics such as Mean-Time-Between-Failure (MTBF) and Mean-Time-To-Repair (MTTR), the probability for a node (denoted as N) to be in a failure state is derived as $P_f(N) = \frac{MTTR(N)}{MTBF(N)+MTTR(N)}$. Based on such probabilities for the nodes of a network, it is consequently possible to derive the probabilities for network paths and correspondingly for traffic flows to be in a failure state. Furthermore, it is possible to assess the risks for the successful delivery of traffic and proactively undertake actions, in order to back-up and remediate potential shortcomings. Such actions can be given by the adaptation of the protection/restoration schemes in GMPLS or by the dynamic adjustment of key parameters (e.g. availability check *Hello* timers) of the employed routing protocols (such as OSPF).

[21] Monitoring information to be delivered by the FDH MonAgent to the FDH SurvAgent for the purpose of Failure-Prediction

7.4.3 Failure Prediction Realization

Having elucidated on various applicable approaches to Failure-Prediction and risk assessment, next the concrete technique which was implemented for the FDH prototype is presented. For the purpose of demonstrating the concept of the Failure-Prediction Functions of the SurvAgent an off-the-shelf algorithm for predicting future values of selected key performance indicators was employed within the FPF module of the FDH SurvAgent. The algorithm is given by the exponential smoothing [Nat12] method which is a commonly used technique across various domains - e.g. finance.

The following equation 7.14 summarizes the principles upon which the value prediction (for particular KPIs) takes place. Indeed, the weighted sum $s(t)$ (with $t \in N^+$) of a number of values is used to detect a trend in the belonging time series, i.e. $x(1), x(2) \ldots$. Thereby, a parameter $\alpha \in [0, 1]$ determines the decreasing importance/influence of the values back in time. That is, the larger α the faster the influence of the past values decreases with decreasing t.

$$s(t) = \alpha \times x(t-1) + (1-\alpha) \times s(t-1) \tag{7.14}$$

The current FDH prototype embeds the above described technique in a way that different monitored KPI values are received over the FDH MonAgent, and subsequently passed to the exponential smoothing function. In order to receive such monitoring information, the FPF module (of the SurvAgent) is pre-configured as to submit the required monitoring requests to the MonAgent during the FDH initialization phase. In addition, a threshold for the trend of each KPI (the trend would be reached by the exponential smoothing in (7.14)) should be given as configuration to the FPF module. This threshold is used to determine whether a trend in the development of a particular KPI should be considered an indication for a failure in the future, and should be the reason for the generation of an alarm. Such an alarm (if generated by the FPF) is then passed to the FDH node repositories and would be considered within the belonging functions/modules of the FaultManAgent or of the SurvAgent (especially the FMF).The corresponding interaction flows were captured in section 6.3.3. Furthermore, the meta-models which provide the foundations for configuring the FPF, as implemented within the current FDH prototype, are captured in section 9.2.3.

7.5 Policy Handling

The current section points briefly to various aspects of policy handling, which are fundamental for the implementation of key FDH functions. Thereby, the policy handling components - as referred to in this section - instrument the triggering of actions/reactions according to input and context from different FDH processes, within the particular situation in the network/system controlled by FDH. The content of the following subsections is a condensed representation of the policy handling artifacts within the emerging FDH architecture and prototype, whilst at the same time the detailed elaboration takes place in other sections/chapters across the thesis, which are correspondingly pointed to.

7.5.1 Introduction

Various components of the FDH framework rely on so-called Event-Condition-Action (ECA) policies in order to complete certain aspects of the (distributed) FDH control loops. This includes:

1. the FPF and FMF modules of the SurvAgent,
2. the FRF,FIAF and FRAF modules of the FaultManAgent, complemented by the supporting component of the ASAF module, and
3. the Policy Evaluation Component of the MonAgent which embeds the Admission Policies, the Notification Policies as well as the Adaptation Policies for the FDH MonAgent operations

Depending on the needs of the above listed components and the technological platforms that were used for the realization of the belonging FDH agents, different design options and principles were considered. Before taking particular design choices, a look into some established policy based solutions - e.g. from the research and industrial domains - was required, in order to gain a solid basis for the design of the Policy Handling within the FDH prototype. This included the review of policy models and evaluation frameworks such as the *Common Open Policy Service (COPS)* [Her00], *DEN-ng* [Str02] [BDJS10] [Str03a] from the domain of Autonomic Networking, *Ponder* [DDLS01], *Common Information Model (CIM)* [CIM16] of the Distributed Management Task Force (DMTF), *eXtensible Access Control Markup Language (XACML)* [XAC17] of the OASIS [22] consortium,and *iptables* [IP617] - as a Linux based tool for the implementation of packet processing policies. Furthermore, router specific technologies such as *Access Control List (ACL)* [Shi07] or *Policy based Routing (PBR)* [Bra89] were considered, in order to establish a broader view on the policy based aspects with respect to network control and management. Further details on state of the art regarding Policy Handling are provided in section 2.2.6.

7.5.2 Design of the involved Policy Handling Components

The current section summarizes the design of the various policy handlers which are required for realizing the self-healing function of the FDH framework. Thereby, the state-of-the-art analysis, which was previously presented in this thesis, played a crucial role while making key design choices. In that sense, the FDH policy handling is specific to the concrete tasks - e.g. Fault-Mitigation within the Survivability Agent or Admission Control in the scope of the FDH MonAgent. Hence, solutions where developed, which are not as generic and powerful as the previously presented frameworks from the area of Policy based Network Management (see section section 2.2.6), instead being dedicated to the specific task and fulfilling it in high efficiency and in a scalable fashion.

Another important aspect is given by the flow for provisioning the policies, which is outlined in section 6.3 and chapter 9. In general, the required FDH policies [23] are specified on a machine running the corresponding tools (e.g. XML-editors), which allow the specification, pre-deployment evaluation and consistency check (e.g. for conflicting policies) within the policy/reaction specification process. Subsequently, the node-specific policies are stored on a logically centralized FDH *Configuration Server* and downloaded during the initialization phase of an FDH instance within a node (refer to appendix section B.1 for further details). Thus, the policies are evaluated within the network nodes during run time, and the corresponding policy obligations (resulting actions) are eventually executed in case they have been allowed by the run-time synchronization/optimization of the node FDH SelfOptAgent.

[22] Organization for the Advancement of Structured Information Standards

[23] Policies are used in the context of the SurvAgent, the FaultManAgent, and the MonAgent of the FDH instance within a network node.

The above elucidated constellation, leading to policies being evaluated inside the network nodes, is a main difference to the IETF COPS [Her00] architecture that suggests the evaluation of the policies on a remote node (PDP - Policy Decision Point), and proposes the enforcement of the policy (i.e. action execution) on the local node. In the scope of COPS, the PDP can have a larger set of resources (e.g. memory or CPU power), which would enable the efficient evaluation of complex policy conditions. On the contrary, FDH is meant to execute the PDP-function within a device which calls for the design of more simple, targeted and (in parallel) efficient fast and scalable policy evaluation methods (and belonging structures). This is one of the main reasons that influenced the decision to implement proprietary and well-optimized/fine-tuned policy handlers for the belonging FDH tasks. A further reason for designing and implementing proprietary FDH policy solutions on a case-by-case basis is given by the sub-optimal Open Source tool support for the most important policy languages of choice (especially the CIM-Policy Model and CIM-SPL). Hence, the opportunity was taken to derive and use specific models and languages for the needs of the FDH agents. These artifacts are efficiently processed within the FDH agents, thereby producing minimal overhead and scaling up with a growing size of the input parameters (i.e. number of policies) as shown later in this thesis.

7.5.2.1 Policy Handling within the FDH Node Monitoring Agent

The basic (meta-)model for Policy Handling within the FDH Monitoring Agent was already presented in Figure 7.3. It includes a *Monitoring Agent Policy* class which incorporates a *Condition* and a *Decision* sub-component. The different types of policies are correspondingly specifications from the generic *Monitoring Agent Policy* class. These different policy types, which are required for the operation of a node level FDH MonAgent, are given by 1) Notification Policies - responsible for deciding whether a particular measurement should be contained or communicated to the requesting entity (be it a an agent or an OSS-management [24] component), 2) Adaptation Policies - contain and determine the logic regarding how key parameters (e.g. observation period) of a monitoring job should be adapted (based for instance on the dynamics of the monitored data as exemplified in appendix E), and 3) Admission Policies - constitute the logic behind the decision whether a newly arriving monitoring request should be served or not.

Examples of how the above listed policies are realized (within the FDH prototype of this thesis) are given in the chapter elucidating on the implementation aspects (i.e. section 10.2). Thereby, a combination of an XML-format reflecting the above meta-model structure and the Java BeanShell [Bea16] scripting language is used, in order to facilitate the ECA type of reaction within the corresponding steps of the FDH MonAgent processes.

The design and technologies, which were pointed to in the current section, are evaluated in various other parts of the thesis. The applicability of the designs is proven in section 7.1.3 and chapter 13, which present case studies involving the FDH MonAgent as a key component. In addition, section 12.2.1 presents various successful performance measurements involving the evaluation of policies within the FDH MonAgent.

The following section proceeds with the rest of the policy components that play a role for the purpose of distributed self-healing as specified in this thesis.

[24] OSS stands for Operations Support System and is the system that enables among others the management of a network, e.g. serving as an interface to the network operations personnel.

7.5.2.2 Policy Handling within the Survivability and Fault-Management Agents

The Survivability and Fault-Management Agents within an FDH node are the other two components, which require to be equipped with the capability to implement policy based decisions, especially in order to execute actions that are triggered by information analysis results from the processes of Fault-Isolation and Self-Optimization.

Given the fact that various specific reaction policies are required for the operation of the FaultManAgent and the SurvAgent, the approach was followed to come up with a meta-model that is in turn specialized for a concrete task within both key self-healing agents of FDH. The overall picture w.r.t. to reaction policies and belonging meta-model for the FDH FaultManAgent and the FDH SurvAgent [25] is provided later on in chapter 9 and especially in Figure 9.6. This meta-model is one of the aspects considered by the network operations personnel for the configuration of the FDH and its self-healing processes in a network.

The concepts that were presented here are implemented in C/C++ and integrated as a library within the corresponding component, be it the FaulManAgent, the SurvAgent or the ASAF supporting component. Thereby, an XML parser [26] was implemented that reads in the XML descriptions[27] [28] of the provided policies and creates a tree data structure in the device memory, which can be subsequently traversed such that the corresponding decision and resulting action(s) can be obtained. Subsequently, the SelfOptAgent might be addressed (if required) over the corresponding API [29] to allow or disallow an action. Finally, the resulting action (policy obligation) is executed as an external CLI command, which completes the required activity (e.g. Fault- Removal, Fault-Isolation Assessment ...).

[25] The *AssIncFuncPolicies* from the FDH supporting components are also part of this picture.
[26] Xerces [XER17] based
[27] These XML descriptions are composed according to the corresponding meta-models.
[28] Concrete examples of the policy configurations based on XML descriptions are given in chapter 10.
[29] Refer to section 10.5 for a detailed description.

Chapter 8

Realization of the Self-Optimization Function

The current chapter elucidates on the realization of the self-optimization features of FDH. The self-optimization function as a whole was already presented in chapter 4. Based on these abstract considerations and definitions, the next sections present in detail the topic of Optimized Action Synchronization, whilst the rest of the self-optimization features are discussed and related to in the coming general section 8.1, in terms of the modules that realize them. Hence, the following paragraphs elucidate on various considerations and derive algorithmic schemes which fulfill **Req 20** from the requirements chapter - **Req 20** determines key aspect of self-optimization required for the efficient FDH operation.

8.1 General Considerations

As presented in section 4.2, the self-optimization features of FDH are given by the following aspects:

- *Dynamic Adaptation of Monitoring Job Parameters* - these aspects are elucidated in section 7.1.2.1, where it is shown which parameters of the FDH Monitoring Agent can be adapted and what are the means for achieving this adaptation. An example for such an adaptive monitoring behavior is given in Appendix E.
- *Optimization of the Fault-Propagation Model* - it is possible to dynamically adjust various parameters and aspects of the employed Fault-Propagation Model, as already outlined in the general chapter 4 on self-optimization within FDH. The adjustment may include removing particular incidents (i.e. faults, errors, failures, alarms) in case there is none probability for those to occur - e.g. in case a network node was dynamically shutdown and related incidents cannot occur any more - or by dynamically adjusting the propagation probabilities of the underlying statistical model (e.g. Bayesian Network or Markov Chain). For the latter probabilities adjustment, different techniques from the domain of Machine Learning and Statistics can be applied. The application of such dynamic methods for adapting the FPM is naturally placed within the FRAF and FIAF functions. This possible FPM adaptation is additionally complemented by the possibility to automatically analyze the control loop execution logs, including the judgments (by the Fault- Isolation Assessment Functions) on

whether the Fault-Isolation process was successful or not. The analysis of these logs allows to extract better Fault-Propagation Models - in terms of propagation probabilities - and store them dynamically within the FPMR repositories of the FDH nodes.

- *Context-aware Self-healing Actions* - the context-aware self-healing actions - as outlined in section 4.2 - are implemented by the different types of policies, which can be embedded in the various FDH agents, e.g. Fault-Removal policies of the FaultManAgent. Thereby, these policies and resulting actions have access to all the information/data within the FDH repositories, which includes node related information in the first place, but also network level information given that the event dissemination mechanisms/components of the FDH have converged with respect to a common view on related events and alarm/incidents. Furthermore, the policies and resulting actions have direct access (over a corresponding interface) to the FDH node local repository for logging control loop executions (Control Loop History Log, i.e. CLH). By accessing this history, each policy can embed a logic as to judge on whether, or in which form, the resulting action should be executed. For instance, as described in section 4.2, the subsequent reconfiguring of a network interface card, in the course of the conducted experiments, has led to a crash of its operating system driver such that after several reconfigurations, a restart of the NIC (or even the whole node) was required as to continue the corresponding network operations. The counting of the number of reconfigurations and the resulting NIC/node restart in that case constitute an example of a context and history aware self-healing function.

- *Selection of Optimal Tentative Actions* - the way this function is realized is presented in the next section 8.2, where a belonging optimization problem is defined and an effective way/algorithm for solving this optimization problem is proposed.

- *Avoiding Contradictory Tentative Actions* - the avoidance of contradictory tentative actions is an aspect that is covered in the course of selecting the optimal tentative actions, which are about to be issued by the FDH agents on different levels (e.g. node or network level). Hence, these aspects are also handled in the next section 8.2, whilst describing the derived optimization problem and belonging procedure to solve it.

The above list of FDH self-optimization aspects represents a wrap-up of the various self-optimization features, which are described across the different chapters of this work and also introductory outlined as an abstract self-optimization function in chapter 4. The next section focuses on the realization of one of the main pillars of this thesis, which is given by the self-optimization for the selection and run-time synchronization of optimal tentative actions, which are about to be issued by various FDH agents across multiple parallel control loops.

8.2 Optimized Action Synchronization

The current section describes the main part of the self-optimization function of FDH, which is given by the run-time action synchronization. Thereby, the goal is to synchronize multiple parallel control loops in the course of their execution and to avoid emergent situations, where each control loop optimizes its own goal, but the overall set of executed actions damages the network/system as a whole. This problem is complementary to the task of policy checking before network/system deployment, where the prepared policies are examined regarding different aspects such as contradictions among conditions and action impacts.

The problem of run-time action synchronization addresses the final stage of all involved control loops, where different actions resulting from previous control loop stages (e.g. Fault-Isolation) are already scheduled for execution. These actions have passed all previous pre-defined constraints (e.g. alarm thresholds) and are the result of multiple policy evaluations and computations (e.g. Markov Chain based Fault-Diagnosis) as well as (traffic) monitoring and data storage/retrieval/dissemination activities. In addition, these actions have emerged in the context of the particular unforeseeable timing aspects (e.g. communication delays) within the network/system being controlled and enhanced by FDH self-healing features. Hence, these tentative actions can be considered as those, which formally satisfy all constraints defined in the pre-deployment phase, but bear severe risks given the intrinsic nature, the timing and emergent situations within real network/system operations[1].

8.2.1 Introduction

The current thesis has reviewed a number of approaches to self-management of/for devices, networks and distributed systems in general (see section 2.2.4 and section 2.2.6). All these approaches address differently the problem of synchronization and ensuring the stability of multiple parallel (autonomic) control loops.

As previously discussed, the issue of action synchronization and optimization is vital in the scope of FDH, since in a complex distributed environment and a composite architecture such as FDH, involving several control-loops executing in parallel, there is an inherent challenge to ensure that the behaviors of the involved agents - e.g. Monitoring Agent, Survivability Agent and Fault-Management Agent - are synchronized towards a global goal. That is required in order to facilitate self-optimization and to avoid a situation, whereby each FDH agent is working towards its own goal but, the overall set of actions/policies degrades drastically the performance and dependability of the network/system, which is controlled by FDH with respect to faults, errors, failures and alarms. Such a situation could even result in unwanted oscillations and instabilities in the control loops.

Coming back to the existing options for achieving action synchronization within the initiatives and research works reviewed in this thesis, a number of approaches can be easily identified. For example, since CASCADAS ACEs [AO16] [CAS16] [ea06] operate based on executable plans, it is straight possible for the autonomic system's developer to embed stability constraints in the resulting (collaborative) behaivour(s). The utility function based approach for deriving policies of ANEMA [DAS09] [ea06] suggests that the resulting behaviors will be intrinsically stable and conflict free during the operation of the network. In addition, the EFIPSANS project [efi16b] has investigated possibilities to design collaborative control loops such that they operate in a stable manner [PCS11] [PTC12]. This includes the application of game theory concepts during the design phase and the application of model driven engineering techniques. The latter can be achieved through different tools or combinations of tools (tool chains), which are applied during the design of the corresponding agents such that their control loops are intrinsically stable and synchronized by design. The set of applicable tools includes modeling environments such as GME [GME16], MagicDraw [BBPK13] and EMF [SBPM09],simulation and verification tools such as UPPALL [BSB+16], Simulink [BSB+16] [TCRMDlF16](UPPALL and Simulink have also some significant

[1]The results in this section have been recognized as important for the domain of network and systems management and have been published as one workshop paper [TCP09], one conference paper [KTP+10], and one journal paper [TS14b], reflecting different stages of the development of concepts and algorithms for the FDH self-optimization function.

modeling capabilities) as well as OMNET++ [VH08], and finally some code generation tool. Examples for tool chains or standalone tools enabling the model driven specification and design of autonomic agents are given in [PTC12] as well as in [RFGS10] and [VH12] [Vas09]. Thereby, the approach in [PTC12] is based on the interplay between existing tools facilitated through a model sharing system called ModelBus [HRW09], whilst [RFGS10] and [VH12] [Vas09] aim at defining a modeling language that enables the specification of self-managing entities.

To underline once again, FDH enables self-optimization not only by verifying the employed configurations, behaviors (e.g. policies and actions) and models (e.g. Fault-Propagation Models) prior to an FDH deployment, but also by introducing the concept of run-time action synchronization for multiple parallel control loops, and by providing the required means to realize it[2]. The mathematical optimization problem that lies at the heart of this concept is introduced within the next section. Based on a model about the impact of potential actions on selected key performance indicators (KPIs), as well as the importance of these KPIs, methods and techniques are proposed for the selection of an optimal subset of tentative actions, which are planned for execution by the involved FDH agents. The approach is validated based on a prototype that allows to conduct a representative qualitative performance and scalability analysis of the proposed concepts.

8.2.2 The Action Synchronization Problem

This section formally defines the run-time Action Synchronization Problem (ASP). The flow starts with a short introduction related to the emerging mathematical model and an overview of similar efforts in the past years. Subsequently, a model is derived that allows to mathematically reason about the optimal subset of tentative actions requested for synchronization by the FDH agents at a point in time. This model consists of a set of mathematical objects, such as a utility function to optimize, and a set of constraints represented by corresponding matrices and vectors.

8.2.2.1 General Considerations Related to the Action Synchronization Problem

In a complex multi-agent environment or architecture - such as FDH - involving more than one control loop that need to execute in parallel, the challenge is to ensure that the agents' behaviors are synchronized towards a common goal. That is required in order to avoid a situation whereby each FDH entity is working towards its own goal but the overall set of actions/policies degrades the performance and dependability of the system in question. In order to achieve the aforementioned common goal, an approach is derived, which is based on the concept of an optimization problem and belonging utility function that incorporates the state of the network/system, and must be optimized by selecting the optimal subset of tentative actions, resulting from decisions made by the single FDH entities.

Research related to the application of utility functions and optimization problems in the area of self-* features for automating system/network management has been ongoing since the release of the IBM Autonomic Computing white paper [ibm05]. The idea of Autonomic Management based on goals, for which utility functions are a way to express, was initially investigated in [LDA+05]. [KD07] is one of the pioneer publications discussing the role of utility functions in self-management. It elaborates on the different classes of policies that can be applied, in order

[2]The general novelty - beyond the FDH scope - of this idea is that it allows for autonomic agents, which were not intrinsically designed as to collaborate, to synchronize their tentative actions and work together towards improving the performance and Quality of Service (QoS) of the network or IT infrastructure in question.

to optimize the corresponding utility function and presents a case study carried out in the context of the Tivoli Intelligent Orchestrator [Tiv16]. [BDK+12] constitutes a further development of the above pioneer work, thereby applying some special optimization techniques (e.g. Incremental Utility Elicitation) and evaluating the approach in the scope of data center management. [TK04] describes a two-level utility function based architecture for monitoring and actively optimizing the performance of a Web-Sphere application environment [HHM04]. However, the application of the utility functions is mainly embedded inside the autonomic entities and there is no additional arbiter component that handles synchronization requests from the involved agents, and chooses to allow or disallow specific tentative actions to be executed. Having briefly looked at the application of utility functions to self-management, the next section proceeds with the formal definition of the ASP problem and derives the belonging utility function.

8.2.2.2 Deriving the ASP Problem

Based on ideas from the theory of Optimal Control [HDPT04], an optimization problem is defined that should be solved by an FDH SelfOptAgent whenever a set of actions needs to be synchronized.

Let Q be a set of performance metrics/KPIs, and $|Q| = n \in N^+$. Let W be a set of weights, each of which is assigned to one of the metrics contained in Q and $|W| = n \in N^+$. These weights represent the importance of a single KPI for the overall health of the system, and thus they all should be positive, i.e. $w_i \geq 0$ for $w_i \in W$, $w_i \in R$ and $i \in \{1, \ldots, n\}$.

Considering a particular point in time t_0, the values of the KPIs in Q are represented by a vector $Q(t_0) = (q_1(t_0), \ldots, q_n(t_0))^T \in R^n$. Suppose that, the higher the values of $q_1(t_0), \ldots, q_n(t_0)$ the better the performance of the system (for KPIs which require to be minimized one can take the reciprocal value), then the following expression gives the system fitness at t_0.

$$SF(t_0) = \sum_{i=0}^{n} w_i q_i(t_0) =< w, Q(t_0) > \tag{8.1}$$

Hence, the goal of the FDH mechanisms in the node/device and the network is to maximize *SF(t)* throughout the operation of the system.

Additionally, let A be the set of possible actions that can be potentially issued by all relevant FDH agents and $|A| = M \in N^+$. By $a_j \in A$ with $j \in \{1, \ldots, M\}$ a single action is denoted. Further, the domain relation of an action $d : A \rightarrow \{0, 1\}^n$ is introduced. The relation takes as an input an action and returns a *(0-1)* binary vector, which contains mappings to the metrics the input action influences if executed. Indeed, if the i^{th} component of the vector is *1*, then the i^{th} metric is influenced by the input action, and respectively if the latter is *0*, then the metric is correspondingly not influenced. Putting together the domain relation outputs for all actions as columns in matrix form results in what is defined here as the domain matrix of A.

$$D = (d(a_1), \ldots, d(a_M)) \in \{0, 1\}^{n \times M} \tag{8.2}$$

The last and most important ingredient for formulating the optimization problem is the impact relation of A - $I : Q \times A \rightarrow R$. The output value of *I(i,j)* stands for the impact the j^{th} action has on the i^{th} KPI. Thus, if only action a_j is executed at point in time t_0, then the new value of q_i will be $q_i(t_0) + I(i, j)$.

Based on the above definitions, and assuming that in a particular time slice a total number of $m \in N^+$ ($m \leq M$) actions have been requested for synchronization, the following optimization problem is defined.

$$\max_{p \in \{0,1\}^n} \sum_{i=0}^{n} w_i(q_i(t_0) + \sum_{j=0}^{m} p_j I(i,j)) \quad \text{subject to: } D_m p \leq c \tag{8.3}$$

In (8.3), *p* stands for a *(0-1)* vector, which gives whether a particular action was allowed to execute or not. Indeed, if the j^{th} position of p is 1, then the corresponding action has been selected for execution; otherwise it has to be dropped. Hence, an optimization with respect to *p* is equivalent to selecting the optimal set of actions requested in a particular time frame. Moreover, the matrix D_m is a sub-matrix of the domain matrix *D* of *A* consisting only of the columns representing the domains of the requested actions.

The vector $c \in N^n$ determines the extent, to which actions with overlapping domains are allowed. For example, if *n=4* and $c = (1,1,1,1)^T$, i.e. only four KPIs are considered, then the additional constraint says that only one action influencing a single metric is allowed. In the case of $c = (2,1,1,1)^T$ two actions are allowed that influence the first metric. Hence, the additional constraint can be used to enforce the resolution of conflicts between actions with overlapping domain regions. Rewriting (8.3) in vector-matrix form results in

$$\max_{p \in \{0,1\}^n} w^T I_m p + w^T Q(t_0) \quad \text{subject to: } D_m p \leq c \tag{8.4}$$

where I_m stands for an $n \times m$ real-valued matrix that contains the impact of the requested actions on the considered KPIs. Thus, $I_m(i,j)$ represents the impact of the requested j^{th} action on the i^{th} metric.

Since the term $w^T Q(t_0)$ in (8.3) is just a constant that reflects the current state of the network, it can be dropped, which is very good news because it means that the values of the KPIs are not needed for the optimization. That fact reduces the overhead produced by the FDH self-optimization within the FDH SelfOptAgent, since no interaction with monitoring functions measuring the KPIs is needed. Hence, the final optimization problem takes the following form:

$$\max_{p \in \{0,1\}^n} w^T I_m p \quad \text{subject to: } D_m p \leq c \tag{8.5}$$

The above optimization problem can be interpreted as *"selecting the most appropriate subset of tentative actions such that the change in the state of the system is positively maximized"*.

8.2.3 On the Complexity of ASP

This subsection investigates around the complexity and the challenges that are expected while solving instances of ASP. First of all, ASP is a special case of integer/binary programming which potentially classifies it as an *NP-complete* problem, i.e. a hard to tackle problem which can be only solved by a non-deterministic polynomial-time Turing machine [Her88]. More specifically,

ASP corresponds to a special instance of the thoroughly studied *0-1* multi-constrained knapsack problem (MKP) [KKA17] [LWC+16] [Fre04]. MKP is defined as follows:

$$\max_{p\in\{0,1\}^n} \sum_{i=0}^{n} g_i x_i, \text{ subject to: } \sum_{j=0}^{n} v_{ij} \leq C_i \tag{8.6}$$

with gains (profits) $g_i \geq 0$, volumes (oder weights) $v_{ij} \geq 0$ and capacities $C_i \geq 0$ with respect to the overall volume (or weight) for each knapsack. Thereby, $g_i \in R$, $v_{ij} \in R$, and $C_i \in R$. The difference between ASP and MKP is constituted by the ranges for v_{ij} and C_i, and correspondingly d_{ij} - the elements of D_m, as well as c_i - the elements of the constraint vector c. d_{ij} can take only binary values, and c_i can take only positive integer values according to the definition of ASP. Hence, each ASP instance is an instance of MKP. MKP has drawn the attention of the research community for years due to its wide field of application (e.g. resource and budget planning). It is known to be an *NP-hard* problem [Fre04] with exact algorithms existing for some special cases, as for example described in [GPP12] and [GG67]. However, based on a review of existing options, it can be concluded that no exact algorithm exists for high dimensional special instances as constituted by ASP.

The above considerations show that finding a solution for the run-time action synchronization problem is not an easy task. Therefore, different heuristics and approximation algorithms can be considered. For example, modern solvers - such as GLPK [GLP16] or Coin-OR [COI17] - are quite advanced and would provide a solution that is the best possible based on the underlying optimization algorithm, e.g. branch-and-bound, simulated annealing, tabu search, etc. Moreover, in the next section, an approach is proposed that allows to partially overcome the obstacles arising due to ASP's computational complexity.

8.2.4 Machine Learning Approach to Action Synchronization

In this section, an approach to run-time action synchronization is proposed that allows overcoming the obstacles of inconvenient problem complexity identified in the previous sections. The ASP problem is correspondingly reformulated and a machine learning methodology to handling the uncertain parameters, resulting from the reformulation, is provided.

8.2.4.1 RASP: Relaxation of the ASP Binary Optimization

The reformulation starts with the previously derived optimization problem, and proceeds to relaxing the binary constraints such that the resulting (reformulated) problem belongs to the complexity class *P* [Pap03], i.e. to the problems solvable in polynomial time. That is, the condition $p_i \in \{0, 1\}$ is turned into an inequality: $0 \leq p_i \leq 1$, for $i \in \{1, \dots m\}$. Hence, the new optimization problem is given by:

$$\begin{gathered} \max_{p} w^T I_m p \\ \text{subject to: } D_m p \leq c, \\ 0 \leq p_i \leq 1, i \in \{1, \dots m\} \end{gathered} \tag{8.7}$$

This optimization problem is a linear program (in optimization theory terminology), which constitutes a convex optimization problem. Hence, given the convex nature, there is only one optimum and every local optimal solution is also a global one, which means that the diverse optimization techniques will always improve iteratively the quality of the obtained solution. The above formulation constitutes a problem belonging to the complexity class *P*, i.e. efficient polynomial algorithms exist for solving this problem. The result of this optimization is a vector containing values from the interval [0, 1]. If i^{th} component of this vector is 0, then the corresponding action is disallowed. Correspondingly, if it is 1 then the action should be issued. If an agent, requesting for synchronization, receives as response a value from the interval (0, 1), then it can either execute or drop the action based on an internal threshold θ_i. These thresholds can be supplied by the human experts tweaking the network/system, which is managed by FDH w.r.t. to faults/errors/failures/alarms. For instance, the history of requests for synchronization can be analyzed offline - e.g. by employing statistical and/or machine learning methods - and appropriate thresholds can be extracted using some optimization tools. In the following sub-section such a methodology is proposed and elaborated in detail.

8.2.4.2 Methodology for obtaining Threshold Parameters for RASP

Assuming that during the operation of a particular system, a history of FDH action synchronization requests - coming from the FDH agents - was collected $r := \{r_1, \ldots, r_{tr}\}$, $tr \in N^+$, a methodology is proposed regarding how to obtain thresholds $\{\theta_1, \ldots, \theta_M\}$, such that the involved FDH agents (FaultManAgent, SurvAgent or MonAgent) are able to decide whether to execute an action or not based on the solutions of RASP obtained by the corresponding FDH SelfOptAgent. One can assume that as long as training data is being collected, the system operates based on ASP. The elements of *r* are *0-1* vectors, where a 1 is set in case the action at the corresponding position in the vector was requested for synchronization within the particular request, and a 0 in case the action was not requested. The results of the RASP optimization - based on the requests in *r* - are then given by $X = \{X_1, \ldots, X_{tr}\}$, $tr \in N^+$ whereby $X_i \in [0, 1]^M$. By defining and solving an optimization problem based on the training data in X, one can obtain the thresholds $\{\theta_1, \ldots, \theta_M\}$ which are in turn given to the requesting agents and used in the course of run-time action synchronization. This leads to the following maximization objective:

$$\max_{\theta \in [0,1]} \sum_{i=0}^{tr} w^T I(X_i - \theta) \tag{8.8}$$

(8.8) can be explained as follows: the distances between the threshold values and the RASP results should be optimized in a way that the impact (reflected by these distances) of the corresponding actions on relevant KPIs is positively maximized. (8.8) can be further reformulated as follows:

$$\begin{aligned} \max_{\theta \in [0,1]} \sum_{i=0}^{tr} w^T I(X_i - \theta) &= \max_{\theta \in [0,1]} \left[\sum_{i=0}^{tr} w^T I X_i - \sum_{i=0}^{tr} w^T I \theta \right] \\ &\iff \min_{\theta \in [0,1]} \sum_{i=0}^{tr} w^T I \theta \iff \min_{\theta \in [0,1]} w^T I \theta \end{aligned} \tag{8.9}$$

The final result in (8.9) is based on the fact that the first sum in the rearranged optimization problem is a constant, because it is computed only based on the available training data.

In order to preserve the constraints given in the ASP definition (8.5), the following constraint is added

$$\sum_{j \in I_m(q_k)} \mathbf{1}_{\{X_{i_j} \geq \theta_j\}} \leq c_k \tag{8.10}$$

with $Im(q_k) = \{j \in N : a_j \in A \wedge I_m(k,j) \neq 0\}$ representing the indices of the set of actions that impact KPI q_k.Furthermore X_{i_j} stands for the j^{th} element of the training vector $X_i \in [0,1]^M$, and c_k is the bounding value for k as given in the "subject to" part of (8.5). Hence, the constraint in (8.10) ensures that the sum of all actions influencing q_k should not be larger than the impact limit c_k, which reflected on the training data means that for the training vectors $X = \{X_1, \ldots, X_{tr}\}$ the sum of all actions scheduled for execution and influencing q_k should not exceed the impact limit c_k.

Combining (8.9) and (8.10) results in the following non-linear optimization problem:

$$\min_{\theta \in [0,1]} w^T I\theta \text{ , subject to: } \sum_{j \in I_m(q_k)} \mathbf{1}_{\{X_{i_j} \geq \theta_j\}} \leq c_k \tag{8.11}$$

As a further step towards increasing the solvability of (8.11), one might consider to treat the constraint in a way that it becomes differentiable such that optimization algorithms can be applied that explicitly make use of the gradient. A possible direction to go is to relax the constraint with the indicator function $\mathbf{1}_{\{X_{i_j} \geq \theta_j\}}$ such that the optimization is done with respect to the distribution of all actions influencing q_k within the training data. Hence, an additional random variable X_{*_j} and belonging probability measure P are introduced, which characterize the distribution of the actions influencing q_k within the training data. This yields the following reformulation of the constraint part:

$$\begin{aligned} &\sum_{j \in I_m(q_k)} P(X_{*_j} \geq \theta_j) \leq c_k \\ \Leftrightarrow &\sum_{j \in I_m(q_k)} [1 - P(X_{*_j} \leq \theta_j)] \leq c_k \end{aligned} \tag{8.12}$$

Thereby, as previously mentioned, $P(.)$ on the last line stands for the (cumulative) distribution of X_{*_j}, i.e. the distribution of RASP results (or in that case training values) for the j^{th} action. To become more specific, X_{*_j} stands for a random variable that represents the various values of the j^{th} elements in the training vectors.Thus, in (8.12) $P(.)$ gives the probability for θ_j to be greater than X_{*_j}. Thereby $P(.)$, in terms of a probability model, must be a distribution over the set $[0,1]$. Indeed, a distribution of X_{*_j} can be computed out of the trainings data, e.g. by calculating the maximum likelihood estimators for the targeted/assumed probability model. Combining (8.9) and (8.12) results in the following optimization problem:

$$\min_{\theta \in [0,1]} w^T I\theta \text{ , subject to: } \sum_{j \in I_m(q_k)} [1 - P(X_{*_j} \leq \theta_j)] \leq c_k \tag{8.13}$$

The beta-distribution [Nat12] is a good candidate for $P(.)$, since it generalizes different possible distributions over the $[0,1]$ interval. Hence, (8.13) can be further specified in a way:

$$\min_{\theta\in[0,1]} w^T I\theta \text{ , subject to: } \sum_{j\in I_m(q_k)} [1 - B_{X_{*_j}}(\theta_j, p_j, q_j)] \leq c_k \tag{8.14}$$

The cumulative distribution function (CDF) of the beta distribution, adapted to the current context for a threshold θ_j, is given by:

$$B_{X_{*_j}}(\theta_j, p_j, q_j) = \frac{\int_0^{\theta_j} t^{p_j}(1-t)^{q_j-1}\,dt}{\int_0^1 t^{p_j}(1-t)^{q_j-1}\,dt}, \textit{ with } p_j, q_j > 0 \tag{8.15}$$

The gradient of (8.15) can be also easily calculated by taking into account that the derivative of a CDF is given by the corresponding PDF (probability distribution function). This gradient can be used to improve the quality of the obtained thresholds given that an algorithm is utilized, which can explicitly make used of it.

Based on (8.15) and (8.13), the optimization problem takes the following form:

$$\min_{\theta\in[0,1]} w^T I\theta \text{ , subject to: } \sum_{j\in I_m(q_k)} \left[1 - \frac{\int_0^{\theta_j} t^{p_j}(1-t)^{q_j-1}\,dt}{\int_0^1 t^{p_j}(1-t)^{q_j-1}\,dt}\right] \leq c_k \tag{8.16}$$

(8.15) has two parameters p_j and q_j, which can be estimated from the training data by using the method of moments, which is defined as follows according to [Nat12]:

$$p_j = \bar{X_{*_j}}\left(\frac{\bar{X_{*_j}}(1-\bar{X_{*_j}})}{s^2} - 1\right)$$
$$q_j = (1-\bar{X_{*_j}})\left(\frac{\bar{X_{*_j}}(1-\bar{X_{*_j}})}{s^2} - 1\right)$$

with

$$\bar{X_{*_j}} = \frac{1}{tr}\sum_{k=1}^{tr} X_{k_j}$$
$$s^2 = \frac{1}{tr-1} \times \sum_{k=1}^{tr} (X_{k_j} - \bar{X_{*_j}})^2$$

Hence, by using the proceeding formulas for the method of moments - thereby incorporating the training data - one can proceed to efficiently solving (8.14) and obtaining the thresholds for the belonging actions and correspondingly for the FDH agents.

Based on these considerations, the following sections compare the quality of action synchronization by solving directly the binary optimization problem in (8.5), and by solving the relaxation based on threshold estimations provided by (8.11), and by (8.13) thereby applying (8.15) as a relevant probability measure. In the following, the combination between (8.13) and (8.15) (i.e. (8.16)) is denoted as the *beta-distribution approach*. Next, the overall proposed approach is shortly related to traditional control theory, before the presentation continues with the promised numerical evaluations.

8.2.5 Relating Run-time Action Synchronization to Traditional Control Theory

The action synchronization problem derived in the previous sections aims at ensuring the stability of the multiple FDH control loops running in parallel. Thereby, depending on the particular configurations, policies and deployed models, the control loops may be based on generic models defined for autonomic computing and networking in the past years (e.g. IBM MAPE [ibm05], FOCALE [SAL06], or GANA [MCR+16]) or on control loops as specified within the area of traditional control theory [HDPT04]. Such control loops are based on specific transfer functions and on the state of the system under control, in case of feedback control loops such as the PID (Proportional-Integral-Derivative) controller. Transfer functions can be seen as mathematical artifacts, which describe the required regulative mechanisms based on different parameters such as time or system feedback. Fundamental properties, which the control loop designer tries to achieve by adjusting the parameters of the belonging transfer functions, are the so-called SASO properties - *Stability*, *Accuracy*, *Settling Time* and *Overshoot* [HDPT04]. *Stability* in that case is considered mainly with the oscillations that could be potentially caused by an improperly selected transfer function parameters. *Accuracy* deals with the degree to which particular desired values of the regulated parameters are achieved. The *Settling Time* property can be seen as an indicative for *"how long it takes"* until the effects of the regulative behavior are achieved. Finally, *Overshoot* gives the maximum deviation (from the desired value) which results from the regulative action.

Within the proposed FDH SelfOptAgent and the algorithmic framework it embeds, one would expect that the underlying control loops, which would be placed within the other FDH agents, are designed considering the SASO properties. The impact of the actions, which result from these control loops, should be modeled as considered after the corresponding settling time. That is, it is presumed that the final effect of the action is taken into account when modeling the potential tentative actions and their impact on KPIs. This way the interplay between the proposed approach for the FDH SelfOptAgent and traditional control loops is facilitated.

8.2.6 Experimental Evaluation of the Self-Optimization Quality

In order to evaluate the proposed approach, a set of experiments was designed such that the quality of the action synchronization achieved by the different mechanisms and techniques can be compared [3]. Since there are no such systems deployed in practice, different requests and models of different sizes were simulated for the experiments - including weight vectors, impact matrices, and constraint vectors. This allowed to gain an impression of the performance of the different techniques on a number of models and combinations of tentative actions.

The measurements presented in this section are meant to benchmark the quality of the action synchronization procedure, in terms of achieved utility value and constraint satisfaction as determined by the corresponding constraint vector. This utility value is obtained by solving ASP directly, or by solving the corresponding relaxation - RASP, and applying thresholds for executing or dropping actions.

A set of models with equal numbers of KPIs and potential tentative actions were used. For each model size, 1000 different model instances were taken. Additionally, 1000 action synchronization requests were simulated as training data for each model size, in order to evaluate the machine learning approach from section 8.2.4. Moreover, 100 action synchronization requests were used

[3] Thereby, the prototype implementation presented later on in section 10.5 was used for gaining experiences and supporting the required measurements.

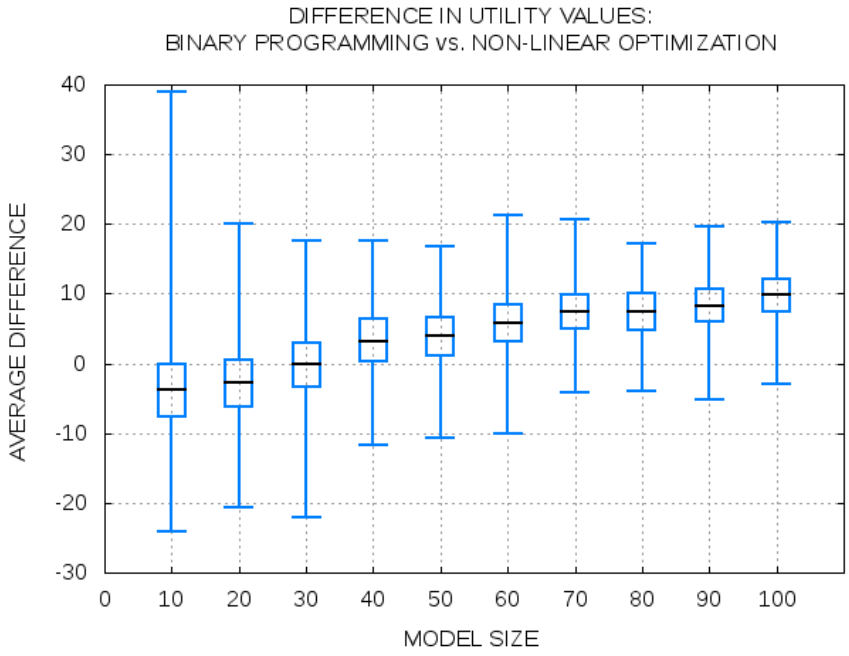

Figure 8.1: The Average Difference between the Utility Values obtained by solving RASP combined with applying Thresholds based on optimizing (8.11), and directly solving the ASP problem

as test data for each model size, upon which the quality of the different techniques was validated. A Matlab/Octave [OCT16] [MAT16] script was developed that implements the machine learning procedures as specified by (8.11), as well as by (8.13) with applying the beta- distribution (8.15) as a probability measure. In the course of this, the NLOPT [NLO16] library for non-linear optimization was utilized. Thereby, initially the "Improved Stochastic Ranking Evolution Strategy" [RY05] was employed, in order to perform a global search for both problems defined in (8.11) and (8.13). Subsequently, a local search was conducted based on the "Constrained Optimization by Linear Approximations" [Pow98] algorithm for (8.11) , and on the "Augmented Lagrangian algorithm" [BM08] for (8.13), in order to find a precise solution locally. Thereby, the algorithm applied for (8.13) can explicitly benefit from the derivatives of the utility function and the constraints, which can be computed based on the beta distribution function. These steps resulted in thresholds obtained based on (8.11) and (or) (8.13) for each model instance. The above introduced test data was used to evaluate the quality of these thresholds in terms of accepting or dropping an action based on a solution for RASP (8.7). That is, firstly RASP was solved, and subsequently by using the corresponding set of thresholds, the tentative actions were accepted or dropped based on the obtained RASP solution vector. The selected actions resulted in a particular value of the utility function, and potentially violated the constraints vector, since the relaxations influence directly the involved constraints. For that reason, on one hand, it is worth to compare the difference between the utility value achieved by using thresholds and the one obtained by solving ASP directly. On the other hand, it is required to measure the magnitude of constraint violation achieved by solving RASP and applying the thresholds. At this point, it should be noted that straight solving the binary program (8.5) does not lead to any constraint violations, since this is the original version of the optimization problem.

Figure 8.1 and Figure 8.2 illustrate the empirical distributions of the differences between the av-

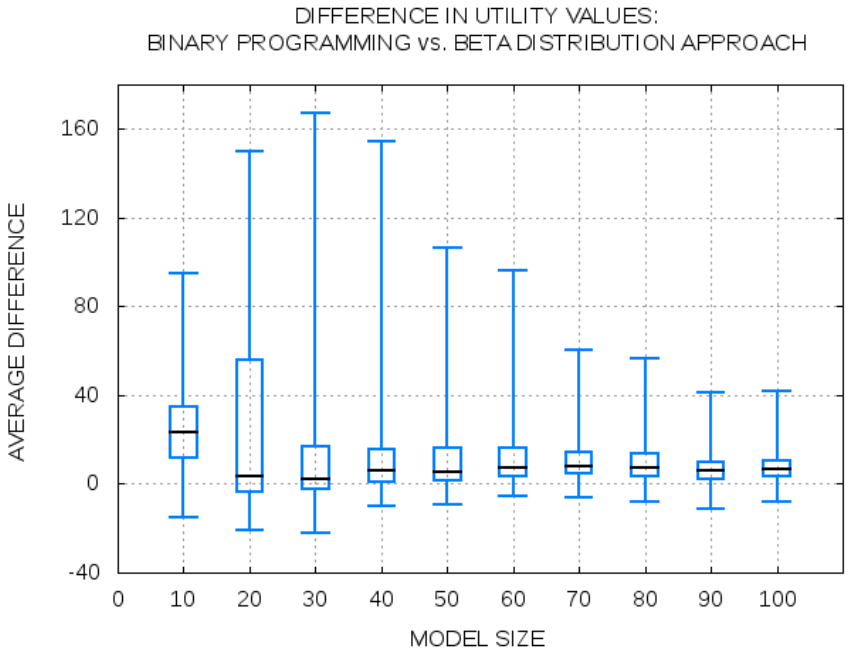

Figure 8.2: The Average Difference between the Utility Values obtained by optimizing RASP combined with applying Thresholds based on the Beta Distribution Approach, and solving directly the ASP problem

eraged utility values of the involved approaches for each model size. The averaged utility values were obtained based on the belonging test data set. That is, the utility values obtained for each action request in the corresponding test data were averaged and presented in the form of the empirical distribution of the difference between these averaged utility values for each of the 1000 instances of the model size. In that case, difference means *"averaged utility value of RASP and (8.11)" - "averaged utility of ASP"* for Figure 8.1, and *"averaged utility value of RASP and (8.13)" - "averaged utility of ASP"* for Figure 8.2. Indeed, a positive value within the empirical distribution means that the corresponding RASP based technique performed better than straight binary optimization. The box plots in the figures visualize the median, the 25% quantile, the 75% quantile, the minimum and maximum value of the experienced sample. Figure 8.1 shows that solving RASP, and applying thresholds obtained by (8.11), performed slightly worse than binary optimization of ASP for smaller model instances. However, with growing model sizes, the RASP based method started performing better eventhough there were still some outliers (in terms of models) for which binary optimization is better. On the other hand, Figure 8.2 shows that solving RASP, and applying thresholds obtained through the beta distribution approach, resulted in better utility values for all evaluated model sizes. Again, there were some outliers (in terms of models) by which the binary optimization seems to be the better approach to obtaining an optimal global reaction for the FDH agents.

The averaged constraint violations while performing RASP based optimization are shown in Figure 8.3 and Figure 8.4. The average value was built based on the above introduced test data set for each model instance. Figure 8.3 and Figure 8.4 clearly indicate that even tough the violations in the case of the beta distribution approach were minor, the non-linear threshold optimization (8.11) clearly outperformed the beta approach when it comes to constraint violations. This is not

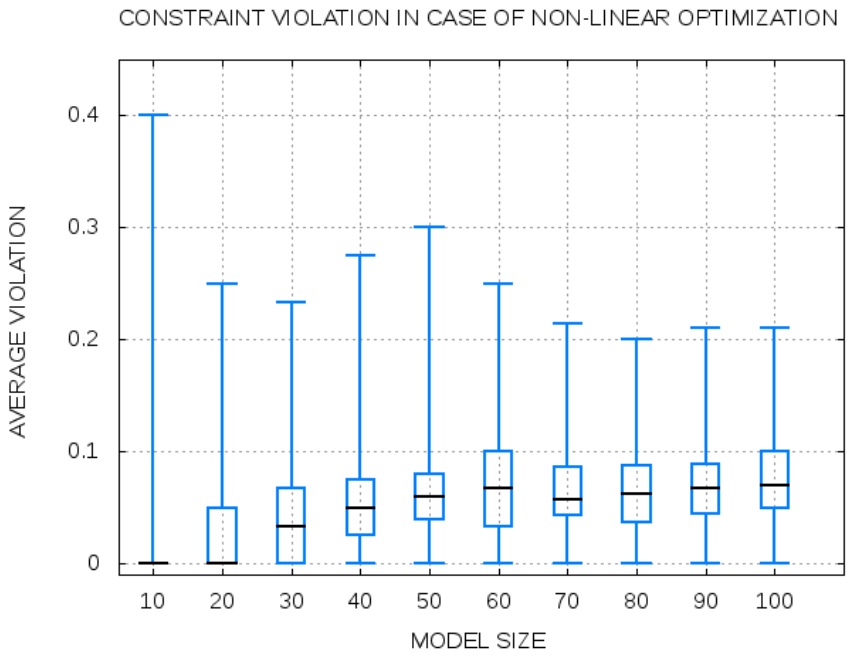

Figure 8.3: The Average Violation in case of optimizing RASP and applying Thresholds obtained through (8.11)

surprising, since (8.11) reflects to a large extent the initial ASP problem regarding the constraints, while the probability based approach (8.13) relaxes them by assuring compliance only a probability distribution base.

In general, it can be observed that the two discussed RASP based techniques performed reasonably as compared to straight ASP binary optimization when it comes to achieved KPI based quality of the overall FDH reaction, and to satisfying the original constraints, which influence the simultaneous execution of actions.

The results presented in the previous paragraphs allow to draw some conclusions and recommendations on how to handle the different aspects of the FDH SelfOptAgent for the purpose of run-time synchronization of parallel FDH control loops. Using RASP as an underlying optimization problem brings some scalability benefits - as presented in detail later on in section 12.2.5 - when it comes to optimizing FDH reactions based on larger models. However, this comes on the costs of some constraint violations, which might be critical in particular computing environments. It remains the task of the FDH configurations' designer to determine whether these constraint violations, in terms of allowing actions with overlapping domains of influence, are acceptable or not. In addition, the thresholds obtained through the beta distribution approach seem to deliver better utility values then those obtained by solving (8.11), when it comes to interpreting the results of a RASP optimization. However, the beta distribution based thresholds imply some higher degree of constraint violation as discussed before. In general, during the experiments, it was observed that given a particular RASP model, it is beneficial to experiment with different available algorithms and parameters for solving (8.11) and (8.13), until the best thresholds are obtained.

With respect to scalability and memory overhead, the conceptual considerations suggest that RASP based action synchronization is the better choice and should be used for time critical and resource constrained systems. This is indeed confirmed by the results presented later on in section 12.2.5

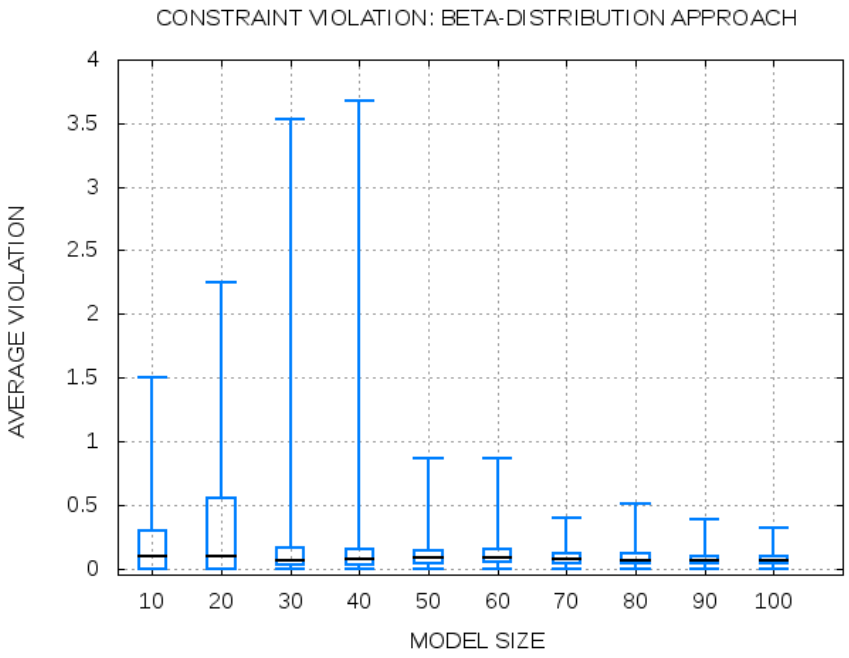

Figure 8.4: The Average Violation in case of optimizing RASP and applying Thresholds obtained through the Beta Distribution based approach

of this thesis.

With these considerations in mind,the next section describes a case study illustrating how the proposed techniques can be applied. Moreover, the next section identifies potential pitfalls, as well as valuable tricks, which should be considered when deploying and configuring an FDH SelfOptAgent for run-time synchronization of parallel FDH control loops.

8.2.7 Example Application

The illustrative case study is a simplified version of the combined case study for the overall framework, which is described in section 13.4. However, the elucidations here focus largely on the self-optimization and action synchronization aspects of the FDH framework. In addition, the involved networking problems were already used for illustrating the operations of the FDH MonAgent (in section 7.1.3) and for exemplifying and evaluating the Fault-Isolation techniques of relevance (in section 7.3.3). In both of these cases, the focus was set to evaluating the properties of the component/algorithm in question, whilst here the main aspect is given by demonstrating the action synchronization/optimization techniques proposed in this thesis.

As previously presented, the FDH FaultManAgent is responsible for realizing self-healing in the long-term operation of the network/system thereby automating the processes of traditional Fault-Management. To recap: traditional Fault- Management is understood [LCZ08] as an interplay of the tasks defined by the ITU-T TMN [ITU00] (Telecommunication Management Network) standard. These tasks are: Fault-Detection - *"detect the presense of fault"*, Fault-Isolation *"find the root cause for the observed faulty condition"*, and finally Fault-Removal - *"remove the identified root cause"*. On the other hand, the resilience of the network depends also on an immediate reaction to an observed faulty condition, which is realized by the FDH SurvAgent. This means that an

immediate action is expected from the FDH SurvAgent, in order to mask the erroneous state until the FDH FaultManAgent mechanisms have managed to remove the underlying root cause(s). Such an immediate action could be for example the result of a regulative mechanism based on a transfer function as implemented in traditional control theory [HDPT04]. Indeed, when it comes to executing the resulting actions, the involved entities - the FDH FaultManAgent and the FDH SurvAgent - need to negotiate, in order to resolve potential conflicts due to multiple Fault-Management or Fault-Masking processes (threads) running in parallel. This negotiation is facilitated by FDH SelfOptAgent within each device and is exemplified in the coming paragraphs.

Figure 7.5 from section 7.1.3 presents the reference network for the case study. The network is basically the same as the one, which was used for previous case studies within the description of the FDH self-healing function, e.g. the Fault-Isolation case study. Each of the routers (*R1* to *R6*) is equipped with a full scale FDH framework including a SurvAgent, FaultManAgend and a SelfOptAgent.

The focus is set on potential faulty conditions around router *R2*, and on potential reactions to these faulty conditions issued by the relevant FDH agents on *R2*. However, presenting the models and mechanisms that drive the reactions of the autonomic entities is beyond the scope of this case study. This mechanisms are described in the corresponding sections and belonging case studies within this thesis.

In Figure 7.5, *R2* is a critical point in the network, since it is a router having different (Gigabit and Fast Ethernet) types of links with different characteristics, as for example MTU (Maximum Transmission Unit) size - 1500 bytes for Fast Ethernet and maximum 9000 bytes for Gigabit Ethernet links. To recap from previous sections: this MTU difference creates the potential for black hole problems as described in [Lah00]. These problems are made possible by the traditional security related suppression of ICMP messages on the router in question having links with different MTUs attached. As previously mentioned, this is especially critical in IPv6 where there is no packet fragmentation on intermediate routers. On the other hand, the (dynamic) suppression of ICMP messages is required in case the network is under attack, since ICMP responses were often used to realize flooding and Deny of Service (DOS) attacks in the past. Hence, it is a good idea to also dynamically regulate the level of ICMP suppression as a reaction to an anticipated attack on the network infrastructure. Thus, the following potential actions, to be issued on R2 by the FDH FaultManAgent agent, are introduced: *set-icmp-suppression-(low/medium/high)*. These actions are expected to differ with regard to the subsets of ICMP message types being suppressed on R2. Furthermore, the problem of link flapping can potentially occur on any of the routers including R2. It is constituted by a link that is continuously going up and down thereby affecting the OSPF routing in the network [WCLL08] (in case OSPF dynamic routing is deployed), such that the routers have difficulties to converge with respect to available routes. An immediate reaction of the FDH SurvAgent to this phenomenon could be given by adjusting the rate of the OSPF *Hello Timer* [Moy98] on the affected routers [WCLL08]. Thus, the following potential actions, to be issued on R2 by the FDH SurvAgent, are considered: *set-hello-rate-(low/medium/high)*. As a result of the belonging automated Fault-Management processes (also of any other kind), the FDH FaultManAgent agent can decide to either restart a Network Interface Cards (NIC) on R2, or to restart R2 as whole. This results in the following potential actions *restart-node* and *restart-NIC*.

Regarding the KPIs to optimize for the case study, the following QoS metrics from the telecommunications domain are considered: *delay*, *jitter*, *packet loss*, *out of order packets*, and *throughput*, as well as the KPIs of *overall security level*, *CPU utilization* and *memory consumption* on R2.

Table 8.1 presents the parameters of the model created based on the case study described hitherto.

Table 8.1: Model Parameters for the Case Study

	restart-node	*restart-nic*	*set-hello-rate-high*	*set-hello-rate-medium*	*set-hello-rate-low*	*set-icmp-suppression-high*	*set-icmp-suppression-medium*	*set-icmp-suppression-low*	*constraints*	*weights*
DELAY	*5*	*4*	*5*	*1*	*-2*	*-3*	*1*	*3*	*2*	*15*
JITTER	*4*	*3*	*5*	*1*	*-2*	*-3*	*1*	*3*	*2*	*10*
OUT OF ORDER	*4*	*3*	*5*	*1*	*-2*	*-3*	*1*	*3*	*2*	*15*
PACKET LOSS	*5*	*3*	*5*	*1*	*-2*	*-4*	*2*	*4*	*2*	*20*
THROUGHPUT	*5*	*4*	*5*	*1*	*-2*	*-4*	*2*	*4*	*2*	*20*
SECURITY LEVEL	*-1*	*0*	*-1*	*1*	*2*	*5*	*1*	*-5*	*2*	*20*
CPU	*5*	*0*	*-4*	*1*	*3*	*0*	*0*	*0*	*2*	*10*
MEM	*5*	*0*	*-4*	*1*	*3*	*0*	*0*	*0*	*2*	*10*
Disallow some of the Actions to execute simultaneously										
	0	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	*0*
	1	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*
	0	*0*	*1*	*1*	*1*	*0*	*0*	*0*	*1*	*0*
Prioritize specific Actions										
	2	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
	2	*0*	*1*	*0*	*0*	*0*	*0*	*0*	*1*	*1*
	2	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*1*
	2	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*1*
	2	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*1*
	2	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*1*	*1*
	2	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*

The first eight columns show the *impact matrix* of the model. The last two columns show correspondingly the constraints and the weights of the model determining the importance of the KPIs to optimize. For the impact of a certain action on the KPIs, integer values from the interval *[-5,5]* are assigned meaning that a negative degrades the KPI, and correspondingly a positive value improves the KPI. For the weights, values from the interval *[0,20]* were considered. A number of interesting aspects are given by the lines, not assigned to a KPI, at the end of Table 8.1. They demonstrate how model parameters can be extended beyond the modeling notions described in section 8.2.2, such that *particular* actions do never get executed simultaneously, and *certain* actions are prioritized. In the current case study, it is logically required that only one action at a time is executed from the sets *set-hello-rate-(low/medium/high)*, *restart-(node/nic)*, and *set-icmp-suppression-(low/medium/high)* in case multiple actions from these sets are requested for synchronization by parallel FDH control loop processes. This is modeled by the second part of Table 8.1, and is achieved through creating a relationship between these actions, and restricting this relationship by extending the impact matrix and the constraints vector. Through this, only one action of each set is selected for execution after a synchronization process. The last segment of Table 8.1 illustrates how an action can be set to have a higher priority than another one. In that case, the action *restart-node* is prioritized over each of the other potential actions. This is achieved by an additional row in the *impact matrix* for each intended prioritization. Indeed within each line for prioritization, *restart- node* is favored based on the new *"impact values"*, and only one of the two actions in question (i.e. *restart-node* and the action with respect to which it is being prioritized) can be selected according to the new *constraints* (refer to the belonging entries in the constraints vector).

To illustrate the operation of an FDH SelfOptAgent based on the case study and belonging model, a prototype was used (described in section 10.5 that handles implementation aspects), in order to issue different requests for action synchronization, and observe the resulting selected subsets of actions. Thereby, the FDH SelfOptAgent was operating on the binary program constituting the ASP version of the above described model. Table 8.2 summarizes some of the results from the conducted experiments in the case study scope. The first row in each segment contains the tentative actions, which were about to be simultaneously executed on R2. The second row in each segment indicates which action(s) has/have been allowed for execution and which disallowed. It can be observed that indeed only one action from the sets *set-hello-rate-(low/medium/high)*, *restart-(node/nic)*, and *set-icmp-suppression-(low/medium/high)* has been selected for execution, in case multiple actions from these sets were requested for synchronization. Moreover, one can also see how the action *restart-node*, when requested, is always prioritized over the others. For the rest, the selection is performed based on the impact of the tentative actions on the utility function of the FDH SelfOptAgent for the network/system in question. This function was defined in (8.3) and (8.5). The results in Table 8.2 show how such an FDH SelfOptAgent can resolve emergent conflicts between parallel FDH control loops and optimize the overall operation of the FDH framework, thereby always trying to achieve an optimal improvement with respect to self-healing.

8.2.8 Discussion and Summary

The above sections introduced a framework for ensuring the conflict-free and synchronized operation of multiple parallel FDH control loops. This framework is accommodated within the FDH SelfOptAgent on node/network level in an FDH controlled network/system. Such a framework is extremely useful, in order to avoid situations when the FDH agents - implementing the parallel control loops - operate in a way as to achieve their own goals, which might unfortunately influence the overall self-healing function in a negative way. In this line of thought, the node/network level

Table 8.2: Illustrative Action Synchronization Requests

REQUESTED	*restart-node*	*restart-nic*	*set-icmp-suppression-low*	
SELECTED	*YES*	*NO*	*NO*	
REQUESTED	*restart-nic*	*set-icmp-suppression-low*	*set-icmp-suppression-medium*	
SELECTED	*YES*	*YES*	*NO*	
REQUESTED	*restart-node*	*set-icmp-suppression-high*	*set-icmp-suppression-low*	*set-icmp-suppression-medium*
SELECTED	*YES*	*NO*	*NO*	*NO*
REQUESTED	*restart-node*	*set-hello-rate-low*	*set-hello-rate-medium*	*set-icmp-suppression-high*
SELECTED	*YES*	*NO*	*NO*	*NO*
REQUESTED	*restart-nic*	*set-hello-rate-medium*	*set-hello-rate-high*	*set-icmp-suppression-medium*
SELECTED	*YES*	*NO*	*YES*	*NO*

FDH SelfOptAgent (responsible for action synchronization and optimization) needs a model that allows it to reason about the impact of different actions such that a synchronization procedure can be performed. Therefore, a mathematical model called ASP (Action Synchronization Problem) was developed, instances of which can be given to an FDH SelfOptAgent, in order to facilitate its operation in a particular context. This model is based on the optimization of key performance indicators (KPIs) relating to the operation of the network/system in question. It allows for: 1) specifying the importance of single KPIs, 2) specifying the impact of the potential actions on the KPIs, and 3) influencing the execution of actions with overlapping impact domains (in terms of KPIs). Moreover, in the course of the presented case study, it was exemplified how the model can be used to: 4) explicitly disallow the simultaneous execution of some actions, and 5) prioritize some actions over others. Unfortunately, this model results in a problem which is of very hard computational complexity. For that reason, an additional effort was presented that aims at relaxing the original problem in a way that it can be efficiently solved. The resulting reformulation delegates some degree of decision to the requesting FDH agents. These decisions are based on parameters - thresholds used to evaluate the numerical solutions of the reformulated ASP - for which a machine learning approach for their configuration was proposed. A comparison between the diverse techniques - direct binary optimization and machine learning based decisions - was also conducted. This comparison shows that the proposed reformulation does not degrade the quality of the action synchronization procedure, while at the same time it has the potential for reducing the memory consumption and improving the response time of an FDH SelfOptAgent (as later shown in section 12.2.5).

The action synchronization and optimization approach proposed here opens a number of exciting research and development challenges. For example, the investigation of further ways for modeling run-time action synchronization and belonging efficient algorithms might be an interesting research topic. Besides, there is a need for sophisticated tooling that allows to easily create ASP models as those proposed here. This could potentially be done in a way that the system's operator does not get involved into the mathematical details, but rather relies on easy to understand atomic artifacts, which can be efficiently combined to enable the FDH framework coping with

various emergent situations. Finally, one might also consider the mapping of multiple parameters of the ASP model to business goals, and using this to achieve revenue by for example adapting the behavior of FDH to specific types of expected traffic or customer specific system needs.

Chapter 9

Network Operations Personnel Perspective

Having presented the distributed architecture for self-healing, as well as the realization of key functionalities in terms of algorithms and processes, the current chapter summarizes and describes the system administrators' view[1] on the FDH framework (addressing **Req 22** from the research requirements chapter). Thereby, it is presumed that the nodes of the network/system in question are all equipped with FDH components. Correspondingly, the system administrator has to "interact" with the FDH components across network in order to:

1. provision the required models to enable FDH based self-healing with self-optimization, and

2. get notified regarding situations, in which the FDH control loops are not able to cope with challenges to the network/system and "give up" thereby escalating to the network operations personnel.

Capitalizing on these aspects, the role of the system/network manager would shift **from** someone, who is involved in the detailed configuration and planning, and who has to continuously deal with the faults across the network in terms of Fault-Management processes, **to** someone who once configures the processes of FDH, observes the operation of FDH and the resilience of the network/system, and improves the operational models of FDH, in order to increase the resilience of the network/system in question.

The presentation proceeds with some general considerations on the network operations personnel view on the FDH nodes within a network or a distributed system. Subsequently, the following section summarizes the configurations required for provisioning the various operational aspects of FDH. Most of these aspects were already presented/touched in the previous chapters. In the current chapter, mainly their role from the perspective of the network/system operations personnel is discussed. Finally, the aspects of FDH involving the network administrators in its operational activities, i.e. human in the loop aspects, are elucidated thereby referring back to the content of previous chapters and shedding a light on the belonging relations.

[1] Initial considerations on the topic presented in this chapter were published in [TC10a].

9.1 General Considerations

Equipping a network with FDH based self-healing mechanisms requires a definition of the way FDH instances interact with the traditional network management infrastructure. This is quite a hot topic since the inbuilt self-healing that is introduced through the FDH principles is expected to transfer some of the management tasks from the network operation personnel to the node/device architectures. However, it is presumed that it is not realistic to design a production-level network with humans completely removed from the overall management processes with respect to faults, errors, failures and alarms - i.e. Fault-Management processes according to TMN.

First, this thesis argues that today's networks need a smooth transition from the "human-driven" Fault-Management to self- healing features as proposed as proposed by FDH. The FDH concepts, algorithms, behaviors and mechanisms will need to evolve over time whilst operating in production networks. Thus, in such a transition period FDH based self-healing systems will still require Fault-Management activities under intensive human supervision.

Secondly, even when the FDH principles and associated mechanisms for a particular network have been extensively refined and matured, the network operation personnel still needs to be kept in the loop as depicted by the "configure/control FDH control loops across the network" flow on Figure 9.1. In such situations, the network administrator acts as a full- time observer and as the highest level of escalation in case the FDH mechanisms across the network encounter a challenge that is beyond their capabilities, and/or should better be escalated to the network management personnel. This requires a number of well-identified criteria in order to determine whether a particular situation should be alerted to the network operation personnel or not. This also means that a history of the occurred incidents and the reactions of the FDH control loops should be made available to the human experts, in order to support the better understanding of processes, trends, and behaviors of the FDH based self-healing network/system. Such information can also be used for preparing statistics that reflect the operation of the self-healing network in terms of utilization and resource consumption.

A high-level view on the information flows between the network management side and an FDH based self-healing network is provided on Figure 9.1. In Figure 9.1, the network management side is represented through a Network Management Console, which constitutes the interface between the operations personnel and the network infrastructure. Thereby, the abstracted information flows to and from the network are represented.

As previously mentioned, the network operations personnel configures the FDH control loops and manages their executions, i.e. besides providing models based on which the FDH instances across the network run their belonging mechanisms, the network administrator closely observes and intervenes - if required - in the operation of the belonging control loops. This includes the reaction of the network managers to faulty conditions, which are beyond the capabilities of the FDH based mechanisms, and for which the FDH instances would raise an alarm according to the escalation criteria presented in section 6.4.

As a further aspect, Figure 9.1 depicts the flow in which the history of control loop executions is conveyed to the network operations personnel, such that it can be analyzed in order to improve the FDH operational models in the long term. Based on the elaborations from Figure 9.1, the following sections proceed with structuring the system administrators' view on the FDH framework for distributed self-healing.

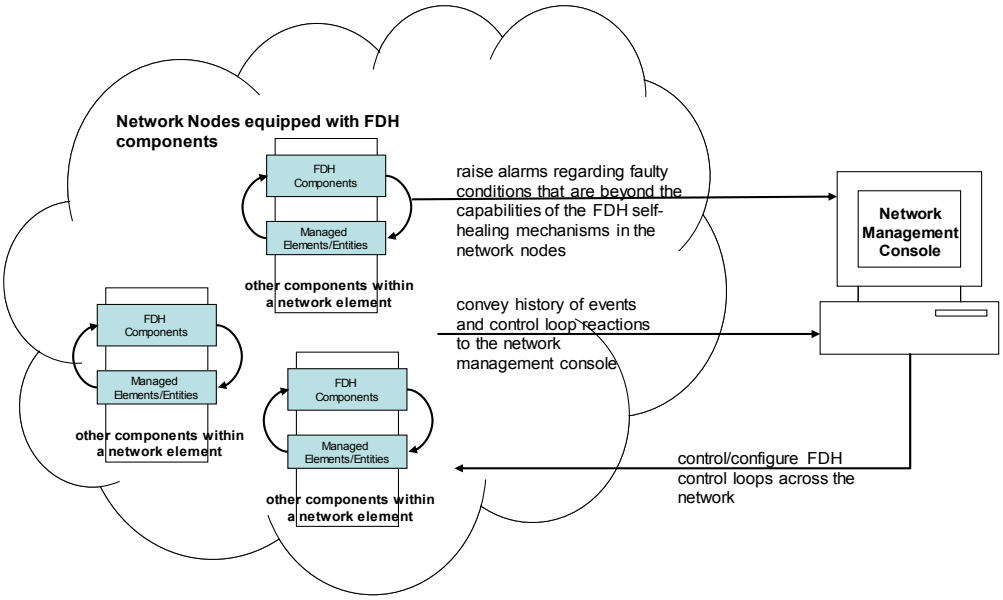

Figure 9.1: High-Level Description of the Information Flows between the Network Management Side and an FDH based Self- healing Network

9.2 Configuration and Administration of FDH

The configuration, control and observation of the FDH control loops across a distributed system or a network require the specification of a number of models, which are elaborated in the following sections. Finally, the involvement of the human in the execution of the self-healing mechanisms is discussed, i.e. the observation of the control loops operations and the involvement of the human in particular Fault-Removal reactions.

9.2.1 Provisioning of FDH Configurations

At the bootstrapping phase of each FDH instance, it is required that particular models are downloaded to the node/device in question and setup in a way that they can be used for self-healing during the operation of the belonging system/ network. These models would be typically provided by human experts configuring and tweaking the processes and self- healing mechanisms of the framework. The models are elaborated in the coming sections and would be created by human experts thereby using special tools. Subsequently, the created models are provided to a centralized Configuration Server that is accessed by the FDH instances, in order to download the models required for their operation. Different protocols can be used for the communication between the FDH instances and the centralized Configuration Server. These protocols can be for example DHCP (or DHCPv6) [Dro97] [Bou03], TR-69 [HBCS11] - a protocol which is widely used in recent years for the configuration of home routers, as well as research prototypes such as ONIX (Overlay for Information eXchange) [TPS+12]. In the scope of the current thesis a simple informa-

tion exchange on top of TCP sockets was realized, in order to enable the download of the models to the bootstrapping FDH instances. The interactions sequences between the bootstrapping FDH components within an instance and the centralized Configuration Server are specified in detail in appendix section B.1. These include the download of FDH operational models such as the Fault-Propagation Model, various reaction policies (e.g. Fault-Mitigation and Fault-Removal) for the FDH agents, configuration models including IP addresses for the exchange of alarm/incident information among nodes, as well as models for self-optimization of tentative actions. More details on initialization procedure for an FDH instance, including all models and configurations which are downloaded to a node, can be found in section B.1 within the appendix.

9.2.2 Monitoring Agent Configurations

The FDH MonAgent within each node requires a set of configurations, which need to be provided by the human experts tweaking the self-healing processes across the network/system in question. The configurations are comprised by the set of policies determining the operations of an FDH Monitoring Agent within a node, and the monitoring requests for the different FDH agents, which are submitted by those agents to the MonAgent.

Given that the realization of the FDH MonAgent is more structurally and architecturally driven, the meta-models for the above mentioned configuration aspects were already presented in chapter 8 dealing with the realization of the FDH self-healing function. Thereby, the corresponding policies meta-model was specified in section 7.1.2.1, whilst the meta-model for monitoring requests was discussed in section 7.1.2.2.

All the configurations are delivered as XML files - based on the corresponding meta-models - to the FDH node instances across the FDH controlled/managed network/system. Examples for the XML files are given later on in the chapter elaborating on the concrete implementation design of the FDH prototype. The delivery of the configurations takes place according to the interaction flows in in appendix section B.1.

9.2.3 Failure-Prediction Model

The Failure-Prediction Model is at the heart of implementing proactive resilient behaviors in the scope of FDH. It determines the mechanisms used by the SurvAgent to acquire monitoring information, correlate it and anticipate the potential occurrence of an incident in the future.

The Failure-Prediction Model as realized within the current thesis is presented in Figure 9.2. The key concept on Figure 9.2 is constituted by the *FPFConfiguration*. The Failure-Prediction Model for a particular FDH node consists of a list of such *FPFConfigurations*. A Failure-Prediction configuration consists of monitoring requests, on one hand, and of alarm "templates" to be utilized for alarm generation based on the information resulting from the monitoring requests, on the other hand.

The decision on when to generate an alarm, based on the monitoring information, is determined by the employed Failure- Prediction method, which is represented through the *FailurePrediction-Method* class in Figure 9.2. The usage of various Failure-Prediction methods is possible, such as Markov Decision Processes [HTC+16] - enabling predictions for future values of the monitored KPIs, Bayesian Networks, Gaussian Processes - as a means for determining the boundaries for the expected values of a particular process, as well as other methods from the areas of Machine Learning, Knowledge Representation (e.g. ontologies), and Statistics.

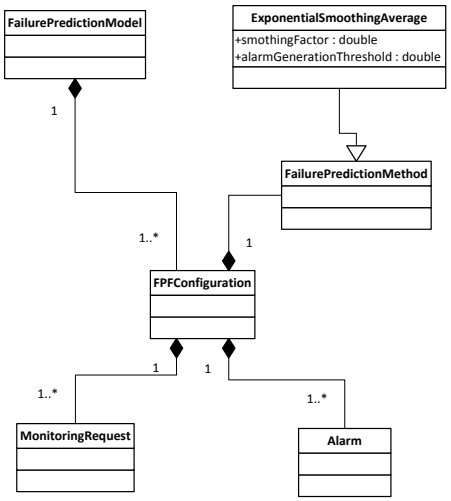

Figure 9.2: UML Class Diagram representing the Failure-Prediction Model of FDH

Within the current thesis, the exponential smoothing average method was utilized for the purpose of generating alarms as a method for Failure-Prediction. More details on the exact algorithm are given in section 7.4.3. The key parameter of the algorithm is constituted by the smoothing factor which is one of the attributes within the *ExponentialSmoothingAverage* class in Figure 9.2. The additional attribute is given by the *alarmGenerationThreshold* that determines the key parameter value which if crossed, would lead to raising of an alarm.

The alarm, which would be generated in case the above described threshold has been crossed, is expected to trigger a reaction of the FMF functions within the SurvAgent. The representation of this alarm within an FPF configuration could also be given by a template type of definition that is automatically filled with the concrete data (e.g. KPI value or crossed threshold) once the particular Failure-Prediction method is about to generate a concrete alarm.

The above presented Failure-Prediction meta-model is used for the definition of the SurvAgent configuration aspects for each single FDH node, i.e. the FPF configurations are node-specific and differ from node to node within a network/ system. These node-specific configurations are downloaded by each device during the FDH bootstrap process, which is depicted by the *getNodeSpecificFailurePredictionModels* and *getNodeSpecificFailurePredictionMonitoringRequests* of the SurvAgent in Figure B.1. Thereby, the downloaded configurations are represented as XML artifacts complying to the Failure-Prediction meta-model. Examples for the XML artifacts are provided later on in the chapter handling concrete implementation related aspects. Finally, the SurvAgent submits the belonging monitoring requests to the FDH MonAgent and starts consuming monitored KPI values, which are utilized by the Failure-Prediction method and lead to the raise of an alarm in case a particular threshold has been crossed.

9.2.4 Fault-Propagation Model

The Fault-Propagation Model - used for performing Fault-Isolation and Root-Cause Analysis within the FDH nodes - is the key operational model for distributed self-healing of the FDH

framework. In contrast to the previously presented configurations, the FPM is defined not on a node-by-node basis, but for the network or distributed system as a whole. This network-wide model is downloaded by the FPMR (Fault-Propagation Model Repository) in each node during the initialization phase of the FDH instance in the device. The corresponding action is described by the *getNetworkWideFaultPropagationModel* interaction in Figure B.3, which shows the download from the centralized FDH Configuration Server.

9.2.4.1 FPM Design Considerations

The design of the Fault-Propagation Model for a particular network or distributed system is within the responsibility of the network/system operations personnel. Thereby, different processes should be considered when initially designing and continuously improving the FPM. Initially, the FPM is designed by the network/system administrators (i.e. human experts) based on their experience, scientific and technical reports describing different potential problems, as well as the anticipation of the human experts regarding various potential problems within the specific network infrastructure. The FPM is generally represented by a graph whose edges stand for different potential events - e.g. faults, errors, failures, alarms, and maintenance events - and whose vertices represent the casual relations between these events in terms of "event A leads to event B".

Having defined an initial Fault-Propagation Model, the network operations personnel provides it to the FDH instances running across the network in question. This model is used in the long-term operation of the network and should be continuously improved based on the history of problems and FDH control loop activations experienced by the network/ system. This history can be analyzed in order to extract additional events/incidents- faults, errors, failures, alarms, maintenance events etc. - that should play a role within the fault-propagation chains of the FPM, as well as to extract new casual relations between events, or to adjust various parameters of these relations, e.g. probabilities in case of a probabilistic approach. A framework addressing these considerations from the network administrator's perspective is presented in the following sections.

9.2.4.2 Ontological based FPM

In general, different representations of a Fault-Propagation Model are required, in order to address the belonging aspects of relevance, including the creation and continuous offline improvement of a an FPM as well as the fast and efficient automated Fault-Isolation within the network devices.

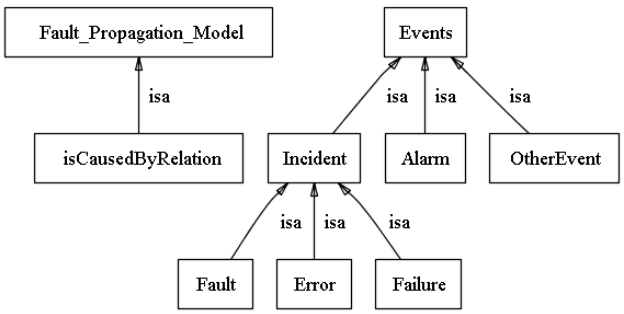

Figure 9.3: Ontology Building Blocks for a Fault-Propagation Model

For the run-time Fault-Isolation within the FDH nodes, a compressed representation of an FPM is considered, which increases the speed of the belonging algorithm (see Markov Chain based Fault-Isolation in section 7.3) and reduces the memory consumption within the nodes at the same time.

With respect to the offline activities, which include the creation of the Fault-Propagation Model as well as its improvement based on the historical data gathered throughout the FDH operation, it is imperative to use frameworks that allow for modeling relationships between events and for capturing semantic relations between the various involved artifacts, such as the values of particular attributes within the modeled concepts. Hence, the use of semantic knowledge representation technologies is required, in order to capture the complex relationships (which can potentially play a role) spanning across different terminologies and different standards.

The building blocks for an ontology based semantic FPM are shown in Figure 9.3. The diagrams there show the semantic relations which build up a Fault-Propagation Model and the belonging events that are structured in the sense of *alarms*, *incidents* and other *events*, whereby the incidents are further specified as *faults*, *errors*, and *failures*. Furthermore, an FPM is built up by a set of *isCausedByRelations* which reflect the various fault-propagation chains among the events on the right hand side in Figure 9.3.

A detailed view on the overall Fault-Propagation Model Ontology is provided in Figure 9.4. It illustrates the above construction of the FPM consisting of *isCausedByRelations*, which include a *causedEvent* and a belonging *cause* instance. The relation between a *cause* and *causedEvent* is enhanced by a *probability* that reflects the likelihood for a particular event to be caused by another event. In addition, each of the events - be it a fault, error, failure, alarm or another type of event - has its own attributes such as descriptions, ids, or threshold information [2] for the alarms.

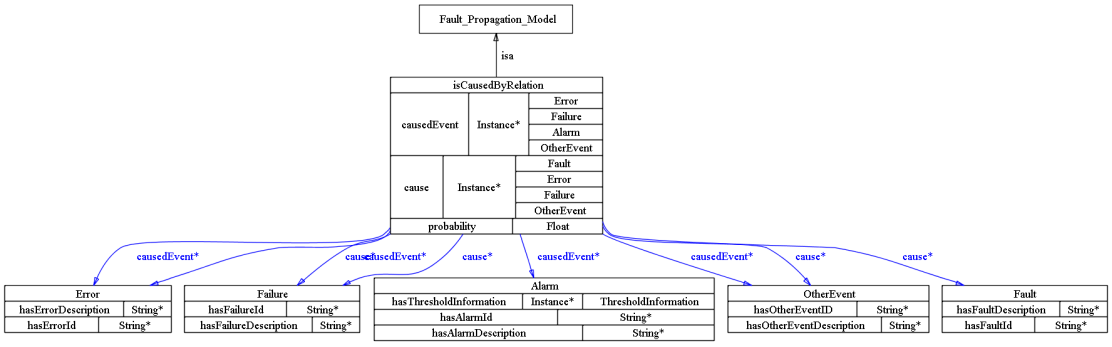

Figure 9.4: Fault-Propagation Model Onotology

Based on such an ontological model, it is possible to add new semantic relations that would facilitate the use of different terminology implied by various communities, standards, or language usage - e.g. "mobile phone" in UK and "cell phone" in the USA. Given that, it would be possible to realize an efficient sharing of FPMs among network operation teams, in order to ease the process of Fault-Propagation Model creation and improvement thereby learning from the experience of colleagues dealing with similar network/system infrastructures. The use of ontological models would also allow the implementation of automated reasoners which would be capable of analyzing different model instances, improving them based on historical records - e.g. improving the proba-

[2] The *ThresholdInformation* description is omitted here, in order not to disturb the flow. In general it consists of a textual description of a particular threshold and a float value, which determines the threshold leading to an alarm.

bilities[3] within the *isCausedByRelations* based on the history of control loop activations within an FDH network, and supporting the human experts tweaking the FDH processes of the network in improving the particular FPM.

9.2.4.3 Framework for FPM Provisioning

Having described the ontological model that was derived in the course of this thesis, the next step is to depict the process that leads to the creation and improvement of the Fault-Propagation Model.

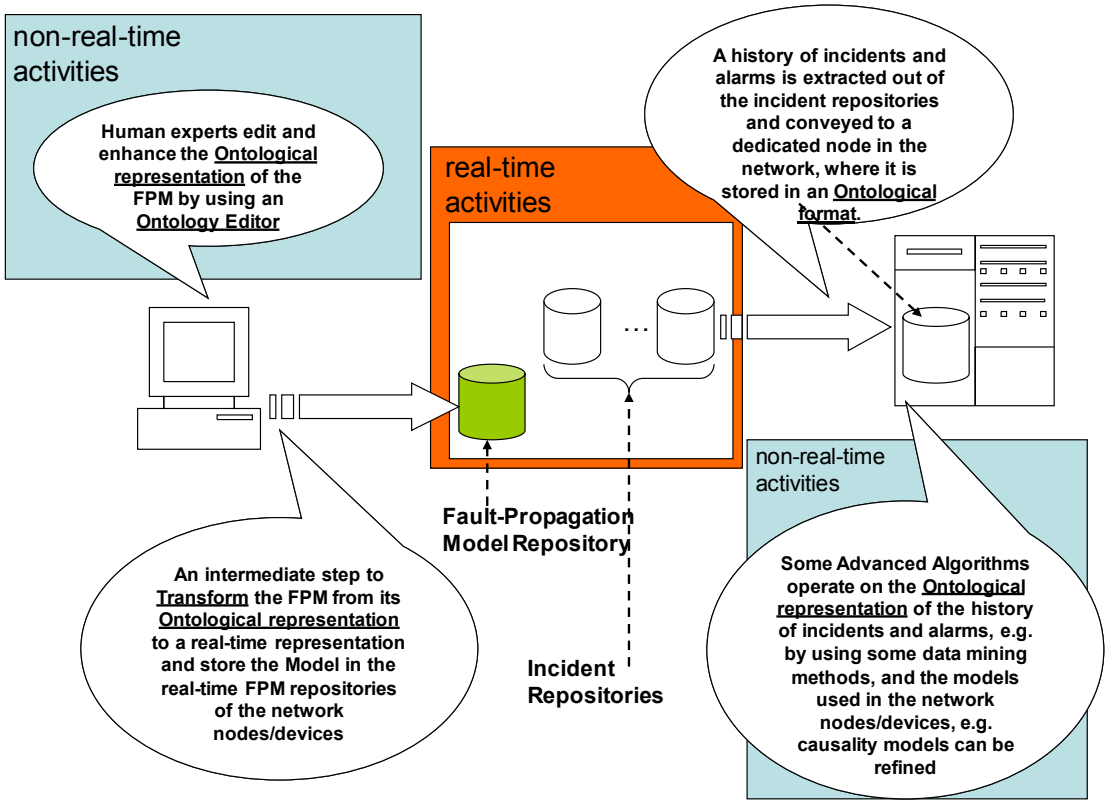

Figure 9.5: Provisioning a Fault-Propagation Model for a particular Network

Figure 9.5 illustrates the view on the human expert management activities related to the Fault-Propagation Model description process. Indeed, a tool-chain is needed that allows the network operation personnel to describe the Fault-Propagation Model for a given network, with a concrete physical topology and software as well as hardware components, and "inject" this Fault-Propagation Model into the node level FPMR repositories of FDH. The description of the Fault-Propagation Model is provided by the network administrator in the above described ontological format using an ontology editor (left side of Figure 9.5). As mentioned above, the usage of ontologies is proposed for several reasons:

1. the semantic knowledge representation constitutes state of the art in the domain of knowledge engineering,

[3] For example by using statistical or machine learning methods.

2. the Fault-Propagation Model knowledge, stored in an ontology format such as OWL or RDF, would allow a number of tools to analyze and query it,
3. the idea of semantic knowledge representation facilitates the bridging of different standards and terminologies,
4. the Protégé [Mus15] [PRO17] editor, together with its graphical plug-ins provides a convenient interface for preparing and maintaining a Fault-Propagation Model,
5. well-proven APIs (e.g. Jena [APA17]) for processing OWL and RDF resources are available.

In addition, the history of events and the judgment on whether the Fault-Isolation process was successful (such judgment can be obtained by the FRAF and the FIAF functions inside the FaultManAgent) can be used to refine the Fault-Propagation Model in the long-term operation of the network. This is illustrated on the right side in Figure 9.5. The history of events can be utilized automatically by some data mining algorithms for the purpose of updating the probabilities of a Bayesian Network or a Markov Chain employed for Fault-Isolation, or can be used by the human experts to gain additional experiences regarding the network and define a new improved Fault-Propagation Model (back to the left side in Figure 9.5).

The question remains of how to ensure the accuracy of a Fault-Propagation Model and facilitate its smooth and efficient construction for a particular network. At the current stage, this is an open research challenge that can be addressed in various ways. For instance, it is possible to have and combine various pre-assessed building blocks for Fault-Propagation Models, i.e. structures that reflect potential fault-propagation chains within a particular topology as well as hardware and software characteristics. These structures can be shared and improved within the networking community, e.g. based on experience and training sets - in the sense of statistical data - gained during network operations. The structures can be subsequently combined based on the concrete network topology and device types.

The above elucidations present the tool chain concept that was worked out in the scope of this thesis and serves the purpose of a vehicle for provisioning Fault-Propagation Models within the FDH prototype. In addition, the opportunities were discussed which are given by using technologies for semantic knowledge representation. The presented approach enables not only the straight creation of FPMs but opens new possibilities for the sharing of sub-structures which reflect fault-propagation chains of events and could be combined, depending on the particular network infrastructure that is managed by the FDH framework with respect to incidents and alarms.

9.2.5 Fault-Removal Functions

The FRF module, part of the FaultManAgent agent, is another component requiring administration. The role of the human expert in that context is to define the reactions of the node-level FaultManAgent to particular isolated faults. Moreover, the network administrator has to identify whether the current Fault-Removal process is allowed to issue particular reaction on its own or should consult an "arbiter", in order to synchronize its tentative actions with those coming from other Fault-Removal or Fault-Masking processes, i.e. whether it should "consult" the SelfOptAgent of FDH or not. Additionally, even if an action has been allowed by the SelfOptAgent to proceed, some Fault-Removal reactions may require the approval of the network operation personnel, e.g. the rebooting of a node.

The above aspects need to be captured in a meta-model that can be used for the configuration of the FDH instances across a network. Thereby the FRF reaction policies are specific for each single device in a distributed system or network. The node-specific FRFs are downloaded during the bootstrapping of an FDH node, which is depicted by the *getNodeSpecificFaultManAgentReactionPolicies* call to the FDH Configuration Server in Figure B.1.

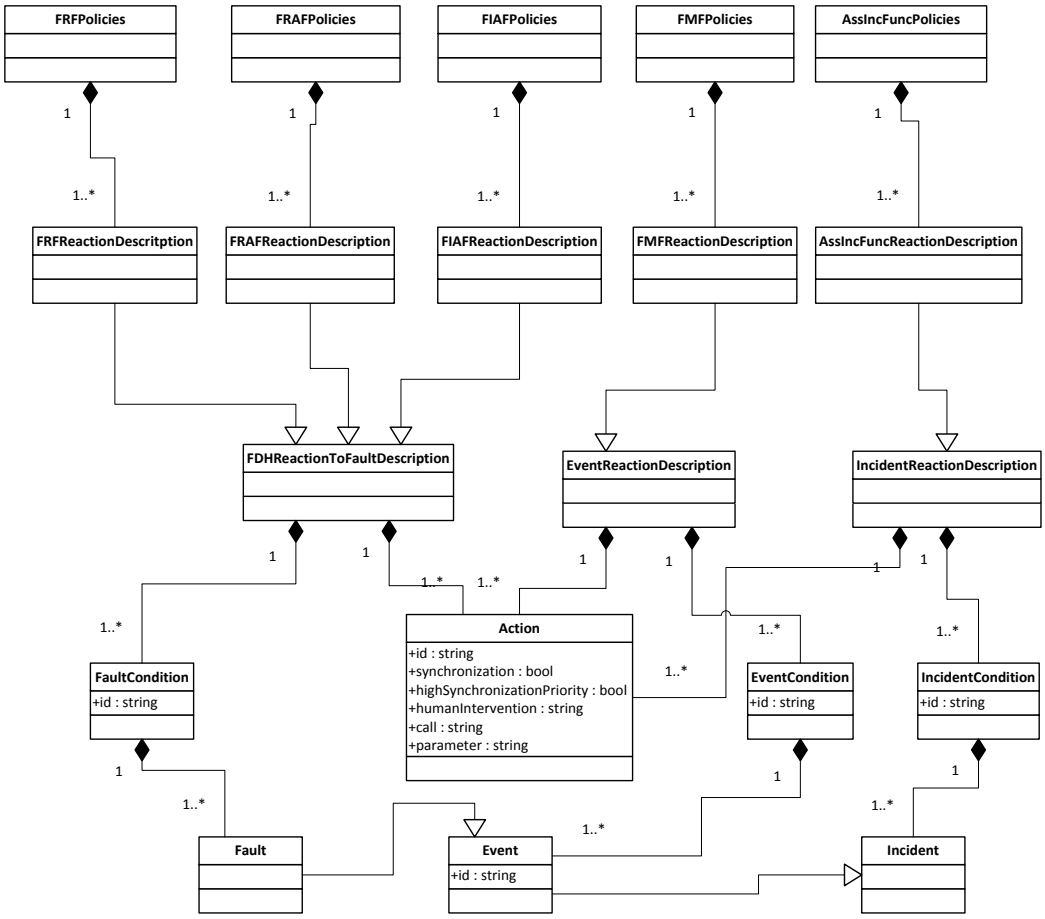

Figure 9.6: The Meta-Model for Reaction Policies within the FDH

Given the fact that various reaction type of policies are required for the operation of FDH, the approach was followed to come up with a generic meta-model for a reaction policy and specialize it based on the demands of the concrete functions. The overall picture w.r.t. to reaction policies and belonging meta-models within the FDH is provided within Figure 9.6. It consists of an *Action* meta-model, which is at the heart of the overall reaction policies description. The *Action* captures the above mentioned attributes which are all relevant when it comes to an FDH agent issuing a reaction. These include in particular an identification string (*id*) for the action and a number of boolean flags which indicate whether the action requires synchronization/self-optimization (attribute named *synchronization*), whether the action should immediately trigger synchronization/self-optimization (attribute named *highSynchronizationPriority*), and finally whether the execution of the action requires the approval of the network operations personnel

(attribute named *humanIntervention*).

Within the meta-model provided in Figure 9.6, the *FRFPolicies* are modeled based on the above described Action model, which is used to compose a generic reaction-to-a-fault-description (*FDHReactionToFaultDescription*) together with a belonging condition (*FaultCondition*) that captures different combinations of faults based on the occurrences of which, one or multiple actions would be issued. A set of specific Fault-Removal Reactions - represented through the *FRFReactionDescription* model on the very left in Figure 9.6 - build up the Fault-Removal policies for an FDH node, which are to be provided on a node-by-node basis in order to ensure the correct reaction of each device to a particular (complex) faulty condition. Thereby, based on the node-specific policies for the FRF part of the FaultManAgent within an FDH node, some devices would react to certain faults (i.e. root causes for erroneous states) whilst others will back-off given the pre-defined FRF policies for each node.

As mentioned above, the FRF node-specific reaction policies are downloaded from the FDH Configuration Server during the initialization phase of an FDH node. Thereby, they are represented as XML files according to the meta-model presented in the previous paragraphs. These XML artifacts emerge as a result of the definition of the node-specific FRF policies by the network operations personnel. Examples of such XML FRF policy representations are given in the implementation chapter (section 10.3) where the concrete prototype designs are elucidated on.

9.2.6 Fault-Mitigation Functions

The Fault-Mitigation Functions of the SurvAgent are meant to realize the reactions of an FDH instance to symptoms and/or indications for faulty conditions in the network. That is, the FMF functions must be able to react to any sort of event, which might be related to an erroneous state and should therefore be implemented by a set of policies, which react to all kinds of events including incidents, alarms as well as generic events, such as maintenance activities etc. In that line of thoughts, the *Action* meta-model from Figure 9.6 is further used for composing the reactions to events (*EventReactionDescription*) based on particular conditions combining generic events - e.g. incidents, alarms, or events in general such as maintenance activities. Thereby, the *EventReactionDescription* constitutes a generalized concept to the specific FMF reaction descriptions (*FMFReactionDescription*) that build up the set of node-specific FMF policies (*FMFPolicies*) to be provided by the network operations personnel for each device in a network.

These node-specific FMF policies are downloaded by the SurvAgent of each node during the bootstrapping phase, which is represented by the *getNodeSpecificFaultMitigationPolicies* interaction in Figure B.1. Thereby, the downloaded configurations are constituted by XML artifacts according to the meta-module descriptions from above. Corresponding examples are given in the chapter handling the prototype implementations.

9.2.7 Asserted Incident Assessment Functions

The AssIncFuncPolicies determine the reactions of the Asserted Incident Assessment Functions, which are meant to check whether a particular incident that has arrived the network - i.e. it was detected by another node and its belonging FDH components - and refers to the local node is indeed present and is not only the result of any emergent network condition (e.g. increased latency). Therefore, the AssIncFuncPolicies should be able to react to incidents be it faults, errors or failures as defined in section 6.6 and section 6.3.2 regarding the dynamic aspects of FDH.

The AssIncFuncPolicies are similarly modeled (see Figure 9.6) as the FRF and FMF policies from the previous sections with the difference that the single meta-models are based on the incident concept - i.e. *Incident*, *IncidentCondition*, and *IncidentReactionDescription*.

The policies for the Asserted Incident Assessment Functions, which are specifically created for each single device by the network/system administrator - based on the meta-model in Figure 9.6, are downloaded by each FDH instance during the bootstrapping phase as described in Figure B.4. The download is represented by the *getNodeSpecificAssIncFuncPolicies* interaction in Figure B.4. Thereby, the FDH nodes download specific XML artifacts, which are defined based on the above meta-models.

9.2.8 Fault-Removal Assessment Functions

The Fault-Removal Functions (FRAF) of the FaultManAgent is another FDH FaultManAgent component on node level that requires a set of policies to be provided by the network operations personnel. The FRAF policies are node-specific depending on the particular faults which can be potentially isolated and should be verified for their occurrence within a particular device.

The FRAF policies for an FDH instance within a node are modeled following the pattern that was presented for the Fault- Removal related policies (see Figure 9.6). Thereby, the *FDHReactionToFaultDecription* constitutes the generalization for the reaction descriptions (*FRAFReactionDescription*) to faults in the context of the FRAF module of the FaultManAgent. Indeed, the FRAF have the task to issue particular actions/commands that verify whether the submitted faults, which were identified as root causes for the current erroneous state, have been indeed removed and the network/system can proceed with its normal operation. In order to capture the overall set of such reaction policies, the reaction descriptions (*FRAFReactionDescription*) are used to compose the overall set of node-specific FRAF policies (*FRAFPolicies*).

After having created the FRAF policies for each FDH node, the network administrator submits them to the (logically) centralized FDH Configuration Server which enables each FDH instance to download its device specific FRAF policies during the initialization phase of the framework. The latter is represented and included within the *getNodeSpecificFaultManAgentReactionPolicies* call to the FDH Configuration Server in Figure B.1. Thereby, XML artifacts are downloaded, which are based on the above meta-model descriptions - corresponding examples are given in the chapter on the prototype implementation aspects.

9.2.9 Fault-Isolation Assessment Functions

In addition to the above presented FRAF and FRF reaction policies for a FaultManAgent, the Fault-Isolation Assessment Functions must be configured as to check whether the results of the Fault-Isolation process are correct with respect to root causes (faults) relating to the local device. This is required in order to avoid following Fault-Removal actions which are based on wrong assumptions regarding the problems to be removed and actions to be issued. Thereby, the FIAF policies are clearly node-specific and address the faults, which reside on or are closely related and can be checked from within a particular network node/device.

The policies for FIAF are modeled in Figure 9.6 and follow the same structure as the FRAF and FRF policies. Thus, an FIAF reaction description (*FIAFReactionDescription* in Figure 9.6) specializes the generic reaction-to-fault description for the FDH (*FDHReactionToFaultDescription* in Figure 9.6). Subsequently, a set of such FIAF reaction descriptions build up the FIAF reaction

policies (*FIAFPolicies* in Figure 9.6) for an FDH node instance, which determine the activities of the FDH FaultManAgent in a node when it comes to locally verifying Fault-Isolation results.

The set of Fault-Isolation Assessment policies is created by the network administrator and stored within the FDH Configuration Server. The node-specific FIAF policies are in turn downloaded to a node during the initialization phase of an FDH instance, which is represented by the the *getNodeSpecificFaultManAgentReactionPolicies* call to the FDH Configuration Server in Figure B.1. Similarly as in the previous subsections, some XML artifacts are downloaded which are based on the above described meta-models.

9.2.10 Event Dissemination Agent Configurations

The configurations of the EvDissAgent contain the list of FDH nodes and IP addresses in general, to which alarm/incident/ event information should be disseminated in the scope of the network, which is managed by FDH with respect to self- healing. In that sense, each node is configured with a list of IP addresses, which might either stand for single network nodes or for a set of nodes, e.g. standard multicast addresses or multicast groups as available in scope of IGMP or MLD.

Besides the pure provisioning of lists of IP addresses, the network operations personnel should analyze the FPM for the network and should determine which particular events, incidents or alarms are of interest for some of the nodes. It might be the case that a node is not concerned with an incident at all, in case the underlying graph of the network-wide FPM does not contain a path that connects the incident with a fault of direct concern for the node itself. In such a situation, an ID of the event/incident/alarm and a list of belonging IP addresses, to which the event/incident/alarm should be sent, are provided by the network operations personnel [4].

The configurations for the EvDissAgent can be either node-specific - for example different lists of IPs for different nodes, or one global configuration can be used for a number or for all of the FDH instances across the network. These configurations are again stored within the logically centralized FDH Configuration Server and are downloaded by the EvDissAgent of each FDH during the bootstrapping phase - represented by the *getListOfAddressesOfInvolvedNetworkNodes* interaction in Figure B.4.

9.2.11 Self-Optimization Model

The Self-Optimization Model is the key configuration model for the SelfOptAgent, which is the main component providing the self-optimization features of the FDH framework. The SelfOptAgent operates based on the algorithmic framework that was developed in the scope of this thesis and is presented in chapter 8. Thereby, a SelfOptAgent is operating in each FDH node, in order to ensure the proper synchronization of the actions undertaken by the FDH components and to achieve the optimization of the self-healing control loops. For that, the SelfOptAgent requires a number of parameters, which are to be provided by the network operations personnel. These parameters represent the various sub-parts which build up the algorithmic framework presented in chapter 8, such as *impact matrix*, *weights determining the significance of KPIs*, and the *constraint* vector determining the number of actions that can simultaneously influence a KPI.

The meta-model that can be used by the network operations personnel is presented in Figure 9.7.

[4] In case there is no specific belonging list of IPs for an event/incident/alarm ID, then it is circulated to the maximum scope which is given by the list of all IPs configured for the EvDissAgents of the FDH instances across the network.

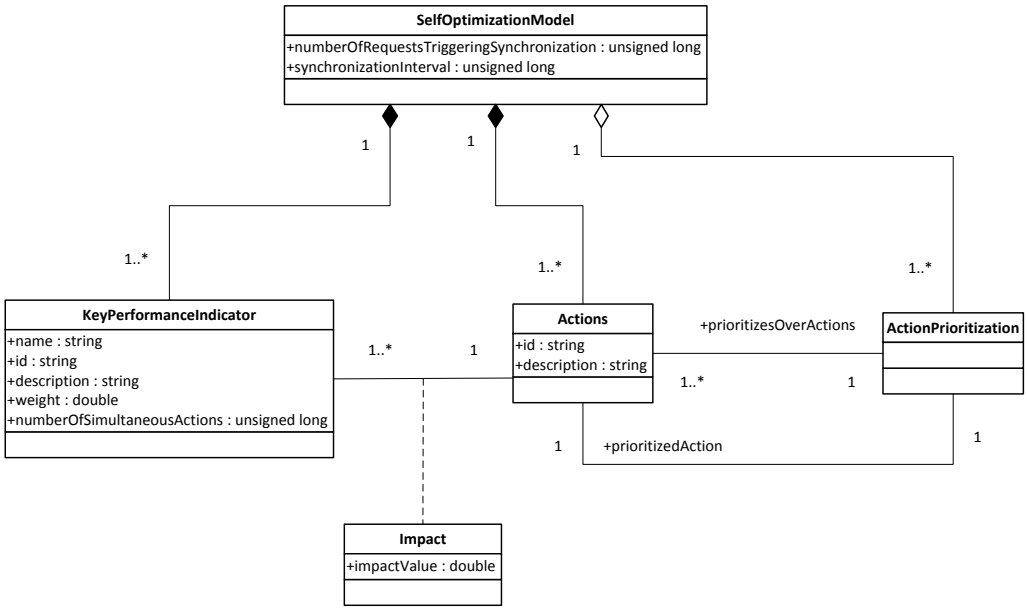

Figure 9.7: The Self-Optimization Model for the SelfOptAgent of FDH

It is build up by key performance indicators (*KeyPerformanceIndicator* in Figure 9.7), action descriptions (*Actions* in Figure 9.7), the impacts (*Impact* in Figure 9.7) of the actions on the key performance indicators as well as the prioritization among the actions. All these aspects flow into the overall self-optimization model (*SelfOptimizationModel*) that contains the number of requests triggering synchronization (attribute *numberOfRequestsTriggerungSynchronization*) as well as the maximum time (*synchronizationInterval* attribute), which the SelfOptAgent should wait before triggering action synchronization/optimization, provided that the number of requests for triggering synchronization was not reached in that period.

As previously mentioned, a number of KPI descriptions are contained within the self-optimization model. A KPI model is specified by a *name*, *id*, *description*, and *weight* - reflects the weight, i.e. importance of the KPI within the self-optimization setting as described in chapter 8. Finally, the number of simultaneous actions allowed to influence the KPI is included (attribute *numberOfSimultaneousActions*).

The above described KPIs are influenced by corresponding actions, which is represented by the impact values between actions and KPIs - modeled through the *Impact* UML association class and the corresponding *impactValue* attribute.

Finally, the actions can be prioritized among each other - modeled through the *ActionPriorization* and the belonging association ends i.e. *prioritizedAction* and *prioritizedOverActions* on the right side in Figure 9.7. This sub-model for action prioritization corresponds to the way some actions were prioritized over others in the scope of the case study for the self-optimization framework in section 8.2.7.

The above described meta-model is used to capture the self-optimization parameters for a SelfOptAgent, either for a specific node or at network-wide level being employed on a device that oversees the FDH activities in the network with respect to self-optimization. In both cases, the

corresponding self-optimization model is defined by the network operations personnel and stored into the FDH Configuration Server. Correspondingly, it is downloaded by the FDH instances during the initialization phase which is represented by the *getOptimizationAndActionSynchronizationModels* models in Figure B.3.

9.3 Human in the Loop

Having presented the various configuration aspects of FDH, it is important to elaborate on the involvement of the network operations personnel within the FDH control loops.

First, the network administrators are responsible for improving the operational models in the long-term operation of FDH and providing these improved models to the running instances. That is, the human experts are involved in a sort of "outer loop". Within this loop, the control loops' logs of the FDH instances are analyzed, and conclusions are drawn regarding what went wrong and which of the models would require what sort of improvement. In order to improve the models, different types of techniques from various domains can be used - for example methods from the area of statistics or machine learning depending on the nature of the underlying model. Concrete examples of the application of such techniques are given by the extraction of the parameters required for self-optimization and action synchronization in section 8.2.4, as well as by the calculation of the Markov Chain transition probabilities in section 7.3.

Besides the above "outer loop", the network operations personnel might be directly involved in the control loops of FDH whilst managing the network/system with respect to faults/errors/failures/alarms. Especially when it comes to the execution of Fault-Removal actions, which change some basic network configuration, it is reasonable to refer to the network administrator in order to give her/him the chance to allow or disallow the tentative reconfiguration. Such an option is also considered within the design of the Fault-Removal reaction policies meta-model in Figure 9.6.

Finally, given that FDH is not able to cope with an erroneous state, an escalation of the particular situation/state to the network operations personnel is required. The escalation takes place based on the criteria for escalating faulty conditions, which were formulated as requirements towards FDH in section 6.4. Consequently, the human experts would need to proceed with manual Fault-Management procedures, in order to restore normal network operations. Having recovered the network/system, the human experts would need to analyze in detail the faulty condition that was escalated and improve the belonging FDH operational models, such that FDH has increased chances to handle similar erroneous states in the future.

Chapter 10

Prototype Implementation and Integration

The current chapter aims at describing the implementation of the FDH prototype, which is one of the main means for validating the proposed concepts, including components, interaction flows, dissemination techniques and algorithms. The software, which emerged in the course of this thesis, is a a research prototype that was used for evaluating the implementability as well as the performance and capability of FDH to solve different network/system problems. Furthermore, the research prototype was used to measure the overhead - an FDH deployment would lead to - within the network nodes. Such an overhead might have an implication on the dimensioning of the network nodes' hardware and might also mean additional costs/expences. All these evaluations are presented in the coming chapters, whilst the sections of this chapter exemplify on how the FDH components can be implemented and integrated as a whole.

The presentation starts with some general descriptions on the choices of technologies for the implementation and proceeds with each of the key FDH components. Thereby, various integration aspects are being elucidated, e.g. the realized APIs that enable the communication between the different modules of the FDH prototype.

10.1 General Description of the Prototype and Overall Component Integration

In order to validate the proposed FDH based approach to self-healing, the components of FDH were implemented using *C/C++* and *Java* on a Linux platform, such that they can easily get deployed on the Linux based routers inside the testbed that was developed alongside this thesis[1].Indeed, besides the Orchestration Engine of the FDH Monitoring Agent no other FDH components were implemented in Java, which means that the rest was realized by means of C/C++. Furthermore, the communication between the FDH components within a device (router, server, media gateway, host ...) was realized using Unix Domain Sockets [SFR03] whereby the corresponding Linux based *C* library was utilized. Thereby, TLV (Type-Length-Value) binary messages were utilized, in order to efficiently exchange data between the different FDH agents and supporting components within a device. The only component that is accessed by means of shared memory

[1]The testbed is presented later on in chapter 11 illustrated in Figure 11.1.

(using the belonging boost *C/C++* library [Sch11]) is the Fault-Propagation Model Repository (FPMR), which is one of the FDH supporting components. Thereby, the FaultManAgent on a node is able to access the FPMR and use the stored Fault-Propagation Model for the purpose of Fault-Isolation. Finally, the communication between FDH nodes is realized by the EvDissAgent in each node. The Event Dissemination Agents exchange data using the UDP protocol.

Having drafted the main technological pillars for the realization and integration of the FDH components on node and network level, the following sections elaborate on each of the key components in turn.

10.2 FDH Monitoring Agent

In this section, the prototype of the above proposed FDH MonAgent is presented. This prototype allowed to iteratively refine the initial ideas of how such a monitoring framework should look. The lessons learned during the implementation and experimentation phase facilitated the clarification of many aspects with respect to the required components and interaction flows. Indeed, section 6.2.1.1 and section 7.1 presented the final version of the FDH MonAgent, after the refinements implied by the prototyping.

In the course of presenting the prototype, the structure of the interaction flows presented in section 6.3 is followed - i.e. with respect to the sequence diagrams in Figure B.2 and Figure 6.8. That is, the prototype presentation starts at the point of the arrival of a monitoring request and elucidates on the way it would finally result in a monitoring job. Subsequently, the processing of monitoring information being initiated within a plug-in is described. Thereby, at certain points, the presentation flow dives into the technical details of the realization, e.g. details around the adopted policy engine technology.

The FDH MonAgent prototype is implemented in Java and can run in the control plane of a router thereby orchestrating different tools for obtaining information regarding traffic and node characteristics. The implemented FDH MonAgent awaits monitoring requests on a TCP socket. The requests are submitted in XML form, which reflects the meta-model presented in Figure 7.4 (section 7.1 in the chapter on the self-healing function realization) and are correspondingly aligned to some aspects of the Network Resource Model [3GP13] proposed by 3GPP. These XML requests are provided (to the requesting FDH agents, i.e. FaultManAgent and SurvAgent) by the network operations personnel following the above described meta-model for monitoring requests. An example for such an XML request based on the meta-model in Figure 7.4 is given in the following listing.

```
<?xml version="1.0" encoding="ASCII"?>
<SubNetwork id="1" type="NIC_STATUS_MONITORING" modifier="create">
  <Router id="1" modifier="create">
    <DeviceInterface id="1" modifier="create">
    <IPv6Address>2001:638:806:66:224:e8ff:fed7:5b2d</IPv6Address>
    <deviceIdentifier>/SubNetwork−1/Router−1/DeviceInterface−1/</deviceIdentifier>
  </Router>
  ...
  <NicStatusMonintoring id="1">
    <granularityPeriod>3</granularityPeriod>
    <reportingIntervals>12</reportingIntervals>
    <thresholdName>tx_dropped</thresholdName>
    <thresholdValue>20</thresholdValue>
    <deviceIdentifier>/SubNetwork−1/Router−1/DeviceInterface−1/</deviceIdentifier>
```

```
    <monitoringIntervalDuration>5</monitoringIntervalDuration>
  </NicStatusMonintoring>
  ...
</SubNetwork>
```

The above listing constitutes an example of a request for monitoring the status of a network interface card (NIC). The UML model behind the possible monitoring requests is presented in section 7.1, which handles the aspects of FDH relating to the realization of a monitoring framework for the purpose of distributed self -healing.

Coming back to the above listed XML (representing a NIC monitoring request), the NIC is specified as belonging to a device (router) and having a specific IPv6 address assigned. Finally, a monitoring job is described that determines the type of KPI - in that case TX dropped packets - to be measured on the NIC in question as well as the threshold value, which would trigger a notification to the requester, in case a particular threshold of TX dropped packets has been crossed within a sampling period.

This presented monitoring request complies to the meta-model in presented in Figure 7.4. Thereby, it should be noted that this meta-model was iteratively extended in the course of realizing different case studies. Hence, all possible artifacts for monitoring requests are summarized in a model which is the common ground for the implementation of diverse monitoring plug-ins and requires to be extended in case new types of monitoring functionality is meant to be provided by an FDH MonAgent.

Proceeding with the implementation and technological aspects of monitoring request processing: Once a request like the above is sent to the Orchestration Engine, the latter uses a DOM [Ind16] parser to prepare an object which is then further processed by the involved modules. Such a DOM object is queried by the different components using the XQuery and XPath [XPA17] [hSJK16] concepts for fetching information from XML. Indeed, the Orchestration engine passes further such a DOM object to the Admission Policies, which have to decide on whether the request can be accepted or not.

At this point, it needs to be elucidated on the way the different policy engines within the framework were implemented/ realized. In the course of implementation, different policy concepts were evaluated with respect to their fast and efficient integration within a Java prototype. More specifically, policy concepts such as those provided within the CIM model [CIM16], SID [SID17a] and Den-ng [Str03a] [Str02] were taken a close look at. Even tough, these models are extremely sophisticated they lack easy to integrate libraries that could be used within the Java based FDH MonAgent prototype. For that reason, a close to Java technology was selected for realizing the policy engines, thereby still allowing for implementing a transformation from CIM, SID or Den-ng policies to the selected format on a later stage.

The choice was made to use the Java BeanShell scripting library [Bea16] which allows for adding and exchanging lightweight scripts, which are interpreted within the Java byte code. These scripts can access the different objects within the Java Virtual Machine, and that way obtain different information and even realize the adaptation of the monitoring plug-ins and services. Concrete example of an FDH MonAgent policy realization utilizing the Java BeanShell is provided later on in this section.

Coming back to the Orchestration Engine processing a monitoring request: After the admission policies have been evaluated, the belonging DOM object is passed to the node/traffic monitoring coordination component where the right plug-in has to be selected. This requires describing the

way plug-ins for the FDH MonAgent prototype are implemented. There exists an abstract plug-in class that defines the following methods which must be implemented by a plug-in:

1. *isAbleToServeMonitoringRequest* - this method receives a DOM object representing the monitoring request and returns true in case the plug-in can serve the request.
2. *serveMonitoringRequest* - this method triggers a monitoring job that is managed by the plug-in and fulfills a monitoring request that it can serve.
3. *stopMonitoringJob* - stops a monitoring job managed by the plug-in.
4. *getMonitoringJobStatus* - returns the status of a monitoring job.

These methods were already presented as part of the abstract *Monitoring Plug-in Interfaces* in Figure 7.1.The first method is used by the monitoring coordination components to select a plug-in able to serve a particular request. The second one is triggered subsequently on the corresponding plug-in, in order to create the required monitoring job.

The monitoring job is then executed within the plug-in based on the parameters submitted in the request, e.g. number of reporting intervals, reporting periods, etc.Moreover, each plug-in can create its own DataAggrCorrFram component which performs a selected sort of aggregation (e.g. mean value) and conveys the aggregated results from the monitoring job to the Orchestration Engine. Subsequently, the Orchestration Engine stores the results in a local MySQL database using the Hibernate persistence framework [Hib17].

At the end of the monitoring information processing loop, the notification and adaptation policies are evaluated and the corresponding decisions executed. The mechanism, these policy modules are based on, was touched on in the previous paragraphs (recall the reference to Java BeanShell scripting library [Bea16]). The next listing presents a simple example of a notification policy. Thereby, the Java BeanShell scripts are wrapped in XML as to obtain a better separation between the preconditions for a policy and the policy logic [2].

```
<?xml version="1.0" encoding="UTF−8"?>
<NotificationPolicy id="1" type="RTT_MONITORING">
  <Condition>true</Condition>
  <Decision>
    import java.lang.Double;
    ...
    if ( Double.parseDouble(policyParameter.getRttvalue())) > 30) {
      return "SEND_NOTIFICATION";
    }
    return "DO_NOT_SEND_NOTIFICATION";
  </Decision>
</NotificationPolicy>
```

As previously mentioned, the core of the FDH MonAgent - i.e. the Orchestration Engine - was developed in Java since it was considered to be not so time critical. This is backed by the assumption that the orchestration part is only going to start some "monitoring sensors" at the bootstrap phase of FDH inside a node, and then let them do the job and directly report detected symptoms to the local alarm/incident repositories. Besides this orchestration part of the FDH MonAgent,

[2]The XML form follows the basic rules specified by the meta-model for policies required by the FDH Monitoring Agent in Figure 7.3 in section 7.1.2.1

all other relevant monitoring entities (especially the monitoring sensors/tools) are implemented in C/C++. For the monitoring components, different freely available monitoring tools - e.g. ethtool [ETH05], tcpdump [ETH05], tcptrace [TCP16] etc. - are wrapped in small C programs that automate the interplay among the tools as to extract the required information, which potentially contains indications for faulty conditions.

10.3 FDH Fault-Management Agent

The FDH Fault-Management Agent is one of key entities of the emerging framework. In general, it is responsible for realizing self-healing in the long-term operation - of the device and the network - by automating the processes of Fault-Detection, Fault-Isolation and Fault-Removal. In order to achieve this, the FDH FaultManAgent implements several modules/components, such as Fault-Isolation Function (FIF) and Fault-Removal Functions (FRF), as well as the belonging assessment/control functions for these processes, namely the Fault-Isolation Assessment Functions (FIAF) and the Fault-Removal Assessment Functions (FRAF).

Similarly, as the other agents and supporting components, the FDH FaultManAgent is realized as a daemon process running on top of the operating systems. The FaultManAgent consumes alarms/incidents/events which are submitted to the local node repositories (part of the supporting functions in section 6.2.3). It receives these events over a Unix Domain Socket [SFR03] communication channel that it has established with the different repositories. Both, the repositories and the Fault-Management Agent are implemented in C/C++ utilizing standard Linux libraries and communication channels such as Unix Domain Sockets and shared memory [Sch11]. Shared memory communication is used in for the process of Fault-Isolation, during which the FDH FaultManAgent has to access the Fault-Propagation Model, in order to diagnose the problems on node level with respect to the current the erroneous state. The Fault-Propagation Model is kept in the Fault-Propagation Model Repository (FPMR), which in turn provides the above mentioned shared memory interface. The shared memory allows the FDH FaultManAgent to directly access the FPM every time Fault-Isolation is required.

With respect to the Fault-Isolation Task: Two different algorithms/schemes were implemented as presented in section 7.3 - a Bayesian Network based Fault-Isolation and a Markov Chain based algorithm [TCC10] that was developed in the course of this thesis. As previously mentioned, the PNL [PNL16] library was used for the Bayesian Network aspects and was correspondingly integrated in the C/C++ prototype. The Markov Chain based algorithm was directly implemented in C++, as part of the FDH FaultManAgent.

For the FRF, FRAF, and FIAF components which all require a policy handler to encode their reaction patterns, a proprietary policy handler was implemented in *C/C++* that allows to supply reaction patterns in an XML format. The XML format is synchronized with the meta-model for FDH reaction policies, which was presented in Figure 9.6. The following paragraphs give concrete examples of such reaction policies in XML.

First, an FRF policy description is exemplified in the following listing. The XML description captures the CLI command (a Linux *ip6tables* call) that needs to be executed, in order to remove the suppression of ICMPv6 messages in Linux router firewall[3]. Thereby, the action to be issued is specified as not needing any human intervention but requiring the synchronization over the FDH SelfOptAgent with other parallel tentative actions.

[3] Details on these circumstances are provided in the various case study descriptions across the thesis.

```
<?xml version="1.0"?>
<FRFPolicies xmlns="http://www.efipsans.org/frf">
  <FRFReactionDescription>

    <FDHReactionToFaultDescription>
      <FaultCondition id="reaction_1_condition_1">
        <fault id="FAULT_Misconfigured_firewall_suppressing_ICMP_packets"/>
      </FaultCondition>

      <Action id="reaction_1_action_1" synchronization="yes" humanIntervention="no">
        <call>/sbin/ip6tables</call>
        <parameter>-D</parameter>
        <parameter>OUTPUT</parameter>
        <parameter>-p</parameter>
        <parameter>icmpv6</parameter>
        <parameter>--icmpv6-type</parameter>
        <parameter>packet-too-big</parameter>
        <parameter>-j</parameter>
        <parameter>DROP</parameter>
      </Action>
    </FDHReactionToFaultDescription>

  </FRFReactionDescription>

</FRFPolicies>
```

Next, an FIAF policy description is exemplified in the listing below. The description is straightforward and captures the execution of a shell script that checks whether a particular firewall rule (suppression of ICMPv6 "Packet too big") is activated.

```
<?xml version="1.0"?>
<FIAFPolicies xmlns="http://www.efipsans.org/frf">

  <FIAFReactionDescription>
    <FDHReactionToFaultDescription>
      <FaultCondition id="reaction_1_condition_1">
        <fault id="FAULT_Misconfigured_firewall_suppressing_ICMP_packets"/>
      </FaultCondition>

      <Action id="reaction_1_action_1">
        <call>/opt/fdh/check_icmp_suppression.sh</call>
        <parameter/>
      </Action>
    </FDHReactionToFaultDescription>
  </FIAFReactionDescription>

</FIAFPolicies>
```

Finally, an FRAD policy description is presented in the below XML example. It constitutes a direct description of a call that starts a Java client, in order to check whether an ICMPv6 black hole was indeed removed. Thereby, the java client can check the firewall rules on a router or simulate a packet flow, in order to check whether the black hole is still present at the critical point in the network.

```
<?xml version="1.0"?>
<FRAFPolicies xmlns="http://www.efipsans.org/fraf">
  <FRAFReactionDescription>
```

```
    <FDHReactionToFaultDescription>
      <FaultCondition id="reaction_1_condition_1">
        <fault id="FAULT_Misconfigured_firewall_suppressing_ICMP_packets"/>
      </FaultCondition>

      <Action id="reaction_1_action_1">
        <call>/usr/bin/java</call>
        <parameter>BlackHolesClient</parameter>
        <parameter>172.19.4.224</parameter>
        <parameter>recover_traffic_after_black_hole</parameter>
      </Action>

     </FDHReactionToFaultDescription>

    </FRAFReactionDescription>

</FRAFPolicies>
```

Having presented the key aspects of the FDH FaultManAgent implementation (including the key utilized libraries, technologies and formats), the next section continues with describing the implementation of the FDH Survivability Agent.

10.4 FDH Survivability Agent

Similarly as the previously presented FaultManAgent, the FDH SurvAgent is implemented in C/C++ and runs as a daemon process on top of Linux in the nodes of the testbed presented later on in Figure 11.1. The Survivability Agent reacts to alarms/incidents/events is receives from the local node repositories, after the FDH MonAgent has submitted those to the local node node repositories. The communication in that case, i.e. from the repositories to the FDH SurvAgent, is technically realized through the use of Unix Domain Sockets as in the case of the FDH FaultManAgent.

10.4.1 Fault-Mitigation Functions

The reaction to the the alarms/incidents/events is realized by Fault-Mitigation Functions (FMF) of the SurvAgent. This reaction constitutes an immediate short-term handling of an erroneous state and is meant to remediate the malfunctioning whilst the FaultManAgent removes the underlying fault (root cause for the erroneous state) in the long-term operation of the network/system. The FMF module functions based on the policy handling mechanisms which were implemented in the course of this thesis. Thereby, a proprietary policy handler was implemented in C/C++ which is also used to evaluate policies within the FMF module and initiate resulting actions. An example of an XML definition for FMF policies is presented in the listing below. These FMF XML artifacts comply to a meta-model presented in Figure 9.6 and supporting the interactions to/from the network operations personnel and correspondingly the management of the FDH framework operating in a distributed system.

```
<?xml version="1.0"?>
<FMFPolicies xmlns="http://www.efipsans.org/fraf">

  <FMFReactionDescription>
```

```
    <EventReactionDescription>
      <EventCondition id="reaction_1_condition_1">
        <event id="BR1_CR1_bad_link_quality"/>
      </EventCondition>

      <Action id="reaction_2_action_1" synchronization="no">
        <call>/sbin/ifconfig</call>
        <parameter>eth2</parameter>
        <parameter>down</parameter>
      </Action>
    </EventReactionDescription>
  </FMFReactionDescription>

</FMFPolicies>
```

10.4.2 Failure Prediction Function

Another key aspect of the SurvAgent is given by the Failure-Prediction Functions, which aim at analyzing different monitored key performance indicators of the system in question, and generating an alarm that indicates an expected faulty condition. The method, which is currently implemented is based on exponential smoothing of measured values and is described in section 7.4.3.

The FPF module is implemented in C/C++ and receives data from "monitoring sensors" (orchestrated by the MonAgent) using Unix Domain Sockets communication. Thereby, the measured values result from monitoring requests, which are submitted by the FPF part of the SurvAgent to the FDH MonAgent. An example of an FPF configuration containing a monitoring request for the MonAgent is given below.

```
<?xml version="1.0" encoding="utf-8"?>
<FPFConfiguration>

  <FailurePredictionMethod>
    <ExponentialSmoothingAverage>
      <alarmGenerationThreshold>40</alarmGenerationThreshold>
      <smothingFactor>0.7</trendParameter>
    </ExponentialSmoothingAverage>
  </FailurePredictionMethod>

  <Alarm>
    <eventType type="COMMUNICATIONS_ALARM_TYPE"/>
    <alarmInformation>
      <notificationId>BR1_CR1_bad_link_quality</notificationId>
      <perceivedSeverity>MAJOR</perceivedSeverity>
      <keywords>
        <keyword>increased</keyword>
        <keyword>round trip delay</keyword>
        <keyword>RTT</keyword>
      </keywords>
      <stateChanged>0</stateChanged>
      <monitoredAttributes>rtt</monitoredAttributes>
      <proposedRepairActions/>
      <corelatedNotifications/>
      <backupId>
        <entityDescription/>
        <entityLocation/>
        <entityInterfaceDescription/>
```

```
      </backupId>
      <thresholdInfo>
        <description>RTT towards a network node</description>
        <!-- Must be the same value as the alarmGenerationThreshold-tag -->
        <thresholdLevel>40</thresholdLevel>
        <locationEntity>
        <entityDescription/>
        <entityLocation/>
        <entityInterfaceDescription/>
        </locationEntity>
      </thresholdInfo>
    </alarmInformation>
  </Alarm>

  <MonitoringRequest>
    <SubNetwork id="1" type="RTT_MONITORING" modifier="create">
      <Router id="1">
        <DeviceInterface id="1"'>
          <IPv6Address>2001:f5a:53:b1c1::b1</IPv6Address>
          <deviceIdentifier>/SubNetwork-1/Router-1/DeviceInterface-1/</deviceIdentifier>
        </DeviceInterface>
      </Router>
      <RTTMonitoringJob id="1" modifier="create">
        <granularityPeriod>2</granularityPeriod>
        <numberOfMonitoringPeriods>10000</numberOfMonitoringPeriods>
        <destination>2001:f5a:53:b1c1::c1</destination>
      </RTTMonitoringJob>
    </SubNetwork>
  </MonitoringRequest>

</FPFConfiguration>
```

This configuration complies to the meta-model presented in Figure 9.2 and defines key aspects, such as the type alarm to be generated in case a particular trend is recognized in the measured monitoring values, as well as the monitoring request to be submitted to the FDH MonAgent for receiving these values. At the start of the SurvAgent, a number of such configurations are imported and belonging POSIX threads started, which handle the required aspects from the *fpfConfiguration*-segments.

Having a more detailed look at the example XML configuration: the *alarmGenerationThreshold* gives a particular threshold value which when crossed - based on calculations from the smoothing average method presented in section 7.4.3 and parametrized with the value of the *smothingFactor*- leads to the generation of an alarm, which is in turn submitted to the relevant local node repositories. The alarm to be generated within this configuration is outlined by the template specified in the tag *Alarm* [4]. Correspondingly, the monitoring request for the FDH MonAgent, which leads to setting up the belonging monitoring job, is specified at the end within the *SubNetwork* tag that follows the meta-model in 7.4 and resembles aspects of the 3GPP Network Resource model as elaborated on in the previous sections. The example monitoring request leads to a periodic RTT (round trip time) measurement to a particular node, which is subsequently analyzed and escalated as an alarm in case an increasing trend is detected, which might be an indication for the deterioration of the network/system performance.

Having described the implementation aspects of the SurvAgent, the next section proceeds with the

[4]The fields for the alarm description are based on considerations from ITU-T standard [ITU92], which were also used within [TC08].

FDH SelfOptAgent, which is the key FDH entity with respect to self-optimization.

10.5 FDH Self-Optimization Agent

In order to analyze the technical feasibility of the proposed self-optimization approach with respect to functionality, scalability and overhead produced by solving the previously derived (see section 8.2) optimization problems in real-time, an FDH SelfOptAgent was implemented on top of Linux in C/C++, and can be started inside an FDH controlled node.Additionally, an API was defined and developed that can be used by "client" components willing to synchronize tentative actions - these components can be any program program running in an FDH node, be it an FDH agent, a MonAgent monitoring plug-in/sensor, or any of the programs/tools implementing an action resulting from one of the policy based components/modules within the FDH architecture.

To wrap up the above considerations: The FDH SelfOptAgent implementation was conducted on a Linux platform using C/C++ and POSIX threads, whilst the "client" components use a specially implemented library for communicating with the node level FDH SelfOptAgent over Unix Domain Sockets.

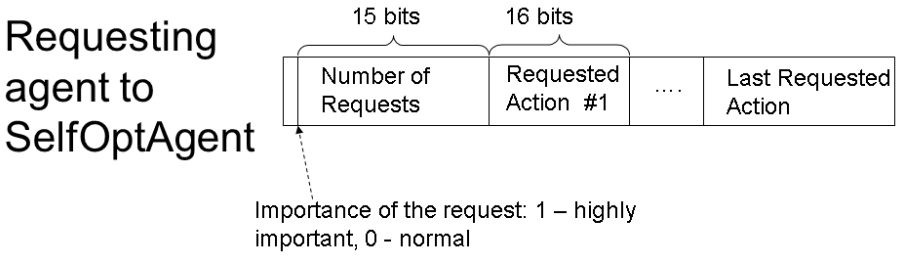

Figure 10.1: A Message carrying an Action Synchronization Request

10.5.1 FDH SelfOptAgent Implementation Architecture

The first issue that has to be addressed, when designing an FDH SelfOptAgent is related to the solver required for solving the ASP and RASP optimization problems (see section 8.2). Research done in the field of electricity spot market optimization problems [TOC06] has compared different open-source solvers for linear and mixed integer programs and gives hints that the SYMPHONY solver (for solving ASP) and the CLP solver (for solving RASP), which are both part of the Coin-OR project [COI17], perform best. Thus, the implementation of FDH SelfOptAgent was built around the solvers provided by the Coin-OR project.

With respect to internal software architecture: the FDH SelfOptAgent consists of two modules - one that takes care of reading and interpreting the configuration data passed to the SelfOptAgent, and a second one that manages the threads serving the "clients".

The configuration data required by an FDH SelfOptAgent component includes:

1. a file containing the *impact matrix*,

2. a file containing the weights corresponding to the metrics' significance,

3. a file containing the *constraints* vector that restricts the number of actions that influence overlapping sets of metrics,

4. the *time interval* after which the solving of the optimization problem is automatically triggered, independently of the number of requests,

5. a *maximum number of requests* that automatically triggers the solving of the optimization problem, and

6. a flag indicating whether ASP or RASP should be solved by the FDH SelfOptAgent, in order to perform run-time action synchronization.

All these parameters need to be supplied to the FDH SelfOptAgent by the time it gets started, and can be provided by a human expert. The aspects of FDH configuration - including belonging meta-models - and the interfaces to the network operations personnel are correspondingly addressed in section 9.2.11.

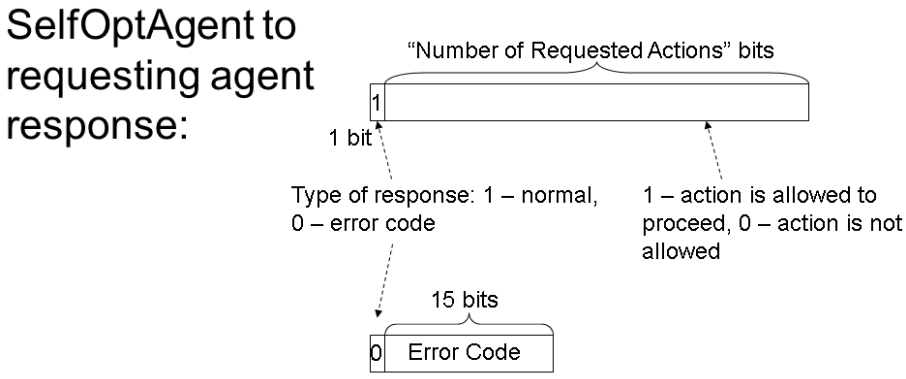

Figure 10.2: Response as a Result of an ASP based Action Synchronization

After the module parsing the parameters has completed the configuration of the FDH SelfOptAgent in a node, a main "server" thread is started that listens for, and accepts "client" connections. In parallel a periodical thread is started that "wakes up" up every X seconds (X is a value provided as a configuration parameter), "gathers" all requests for synchronization, prepares the requests, the impact matrix, the constraints vector as well as further required parameters in the form required by the native interface of the solver in use (SYMPHONY or CLP), and finally invokes it in order to obtain a solution for the current set of tentative actions to synchronize. Consequently, the relevant parts of the solution are communicated back to the corresponding requesting agents. Apart from the periodical task/thread, every time a connection gets accepted by the FDH SelfOptAgent, the requests are stored in a common (for all threads inside the SelfOptAgent-process[5]) registry. In cases that the request was submitted and marked as *highly important*, or the *maximum number* of requests after which synchronization is automatically triggered has been reached, the solving of the optimization is invoked, taking into account all synchronization requests stored in the common registry since the last time optimization was triggered. In order to synchronize all the concurrent threads around the common registry and the solving process, POSIX mutexes [But97] are utilized.

[5]The term *process* denotes a Linux process on top of the operating system.

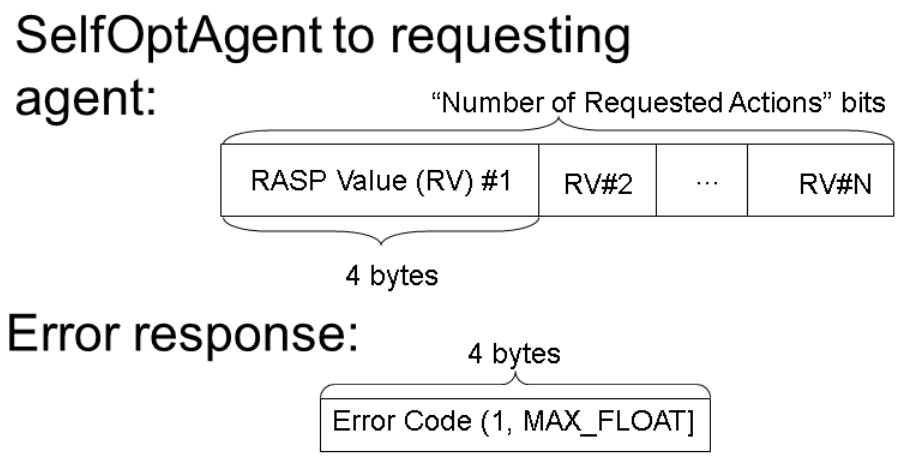

Figure 10.3: Response as a Result of a RASP based Action Synchronization

10.5.2 FDH SelfOptAgent Interfaces

In addition to the above described implementation aspects, the communication between the requesting entities and an FDH SelfOptAgent requires the specification of a message format for the exchange of requests and responses in case of an ASP or RASP solving SelfOptAgent. These messages can be exchanged over Unix Domain Sockets inside a device (as currently done), or over specially designed protocols for control information exchange between nodes, such as the ICMPv6 inspired IP based Generic Control Protocol (IGCP) [BCX+11].

Figure 10.1 presents the format of an action synchronization request message as implemented in the FDH SelfOptAgent prototype developed in the course of this thesis. The first bit is used to indicate the urgency of the request. The following 15 bits are used to encode the number of requested actions, followed by a set of 16 bit integers representing the requested actions (in terms of numerical IDs).

Depending on whether the FDH SelfOptAgent in question is pre-configured to perform an ASP or a RASP optimization, different response formats are required. Figure 10.2 illustrates the message carrying the response from an ASP based action synchronization. The first bit of the message indicates whether the synchronization was successful or not. In case of a failed synchronization, different error codes can be reported within the next 15 bits. In case of a success, a corresponding number of bits is conveyed which reflect the request for action synchronization thereby indicating whether a tentative action is allowed (bit value 1) or disallowed (bit value 0). Additionally, Figure 10.3 depicts the response message in case of RASP synchronization. It consists of a sequence of thresholds each encoded in 4 bytes. These are meant to be floating point numbers in the range between 0 and 1. In case of failed action synchronization, an error code is returned that indicates the type of problem occurred within the FDH SelfOptAgent.

The above presented implementation aspects cover the key component regarding self-optimization within the FDH. The operation of the main FDH agents within a node - i.e. FaultManAgent, SurvAgent and SelfOptAgent - are facilitated by a number of supporting components, the implementation of which is presented in the next subsection.

10.6 FDH Supporting Components

The FDH supporting components encompass a set of repositories and attached functions - i.e. *Node-Specific Event Management Functions and Repositories*, *Monitoring Information Repository*, *Fault-Propagation Model Repository*, *Control Loop History Log* - as well as the *Event Dissemination Agent* that takes care of synchronizing the FDH nodes with respect to alarms/incidents/events.

Regarding the repositories: The node-specific repositories for alarms/incidens/events were implemented as daemons on top of Linux and are based on previous work conducted in the course of [Tch09] and [CTS08]. Thereby, various traditional data structures (e.g. linked lists and different types of hash maps) were efficiently implemented in *C* and used for storing and retrieving alarms/incidents/events within the FDH nodes. In addition, the communication to and from the repositories was realized using Unix Domain Sockets, which means that each FDH component/agent needs to setup a Unix Domain Socket towards the relevant repositories at its start. In the course of that, the repositories act as a server whilst the other FDH components connects as clients over the Unix Domain Socket. Furthermore, it must be mentioned that the FDH node-specific repositories for alarms/incidents/events provide an API that allows random pieces of software running inside a network node to store and query data utilizing the FDH repositories. Hence, random programs/components within a node can act as monitoring sensors or can adapt their behavior with respect to the situation regarding alarms and incidents within a network node.

There is one additional component, which is attached to the FDH node-specific repositories for alarms/incidents/events. This component is given by the *Asserted Incidents Assessment Functions* which is responsible for verifying whether incident information provided by remote FDH nodes (i.e. which has arrived over the overlay of FDH EvDissAgents) and pertains to the local node is indeed valid. Hence, these functions need to implement a policy based reaction which verifies such incident claims/assertions for erroneous states. That is, tiny checks should be issued in terms of monitoring activities that examine the asserted states locally. The component is also implemented in *C/C++* as a Linux daemon and utilizes the proprietary policy engine designed within this thesis. The corresponding policy model are provided in Figure 9.6, which deals with the aspects of FDH configuration and management to be conducted by the network operations personnel. Given that a particular incident/alarm cannot be verified as present within the local node, the *Asserted Incidents Assessment Functions* can use corresponding APIs - of the local FDH EvDissAgent - to disseminate a belonging recovery notification across the distributed system. If an alarm/incident gets confirmed, it is stored in the corresponding repository using the above mentioned API of the alarm/incident/event repositories (see section 6.3.2 for such dynamic aspects).

The supporting component *Monitoring Information Repository* is responsible for keeping general monitoring information, such as the values of different QoS metrics towards particular nodes (e.g. jitter, delay, round-trip-time ...) or the current memory consumption in a network node. This information is obtained by the different monitoring sensors of the FDH MonAgent. The data is kept in a MySQL database [MYS17], which is accessed (mainly by the FDH MonAgent) over a Hibernate [Hib17] persistence layer. That way the SQL database can be easily exchanged depending on the deployment circumstances (e.g. PostgreSQL [Pos17]). Thereby, the data stored in the database can be dynamically extended over the Hibernate mechanisms.

Focusing further on the FPMR: the Fault-Propagation Model Repository keeps the Fault-Propagation Model for facilitating Fault-Isolation in a shared memory segment in each FDH node. Hence, the Fault-Propagation Model can be accessed using *C/C++*-libraries [Sch11] for

accessing shared memory. Indeed, the FDH FaultManAgent utilizes these libraries in order to traverse the Fault-Propagation Model each time Fault- Isolation needs to be performed within the FDH FaultManAgent control loop. Furthermore, given that the FPMR is implemented as a *C/C++* based daemon on top of Linux, it starts a TCP client socket for obtaining/receiving new versions of the Fault-Propagation Model. These new versions of the FPM are generated based on various techniques and processes as outlined in section 7.3. The belonging interaction flows are captured in section B.1 within the appendix.

Moving on to the FDH Event Dissemination Agent, it is also implemented in *C/C++* on top of Linux in each node of the testbed in section 11. For the node internal communication to and from the FDH EvDissAgent - see different calls within the sequence diagrams in section 6.3 - Unix Domain Sockets are utilized as typical for the FDH implementation in the course of this thesis. For the communication among the FDH EvDissAgents, residing in different nodes, UDP was used as a protocol for communicating short datagrams containing small in size alarm/incident/event descriptions, which are TLV based and described in previous work such as [Tch09] and [CTS08].

In order to increase the chance of complete successful alarm/incident/event dissemination between the EvDissAgents (over unreliable UDP datagram messaging), a canonical type of flooding was implemented where the node that has detected/ generated the alarm/incident/event sends the resulting short message (using its EvDissAgent component) to a list of addresses (potentially including multicast addresses) [6]. In turn, every recipient node forwards again the message to all the addresses in question, in case the alarm/incident/event description contained in the message is unknown to it, or drops it in case it has already received it. This procedure was analyzed using the *OMNET++* simulator and the resulting scalability analysis is provided later on in section 12.2.2. Further algorithmic elucidations and discussions were given in section 7.2.

Finally, the *Control Loop History Log* is realized as a text file containing logging information from the control loop executions within a node. This information is generated and can also be parsed and utilized by the small programs (i.e. Fault-Removal or Fault-Masking actions) realizing the final reactions of the relevant FDH control loops - e.g. FaultManAgent or SurvAgent control loops. Given the open and flexible nature of the reactions (i.e. of the belonging software components), no specific format for the Control Loop History Log is specified. This aspect is rather left open to the network engineers/managers implementing the specific reactions for a particular FDH managed/controlled network.

10.7 Interface to the Network Administrator

In the course of prototyping the FDH framework, a number of possibilities were analyzed for implementing a toolkit that meets the needs of the interface between FDH and the network operations personnel. The high-level specification and requirements for this interface and correspondingly for such a toolkit were presented in section 6.2.4 and section 6.3, where the architectural and dynamic aspects of FDH were specified. Regarding the architectural aspects, the key components which were specified are given by the *Bootstrap Manager*, the *Configuration Server*, the *Provisioning Tools* and the *Network Monitoring System* that is used to oversee the operation of FDH in the network as a whole. In the following, the key technological aspects of the implemented components are elucidated.

The current implementation of FDH does not contain a dedicated *Bootstrap Manager* component,

[6] The TLV format of the messages was specified in previous works such as [Tch09] and [CTS08]

which constitutes a deviation from the framework specification in section 6.2.4. Moreover, it can be claimed that the Bootstrap Manager extends into each of the relevant entities, since every FDH entity in a node connects over TCP to the central *Configuration Server* and downloads its relevant configurations, be it for instance a Fault-Propagation Model or any sort of reaction policies. The Configuration Server is implemented in Java and acts as a TCP server utilizing the standard Java-TCP-socket libraries. Upon receiving a connection request from an FDH entity within a node, the correct configuration data to deliver is selected and transmitted to the requesting entity over he current TCP-channel. The interaction flows relating to the Bootstrap Manager and Configuration manager are presented in section B.1 within the appendix.

With respect to the utilized *Provisioning Tools* within the prototype developed in this thesis: The Protégé [PRO17] ontology editor was used for the description and specification of a Fault-Propagation Model for the case studies (e.g. section 7.3.3) within the testbed from section 11. The Fault-Propagation Model is based on the concepts described in section 7.3 and proposed in [TCC10]. Furthermore, Protégé allows for visualizing the Fault-Propagation Model structure by using diverse plug-ins as this was exemplified in section 9.2.4.2, dealing with the processes and models for FDH management from the perspective of the network administrator. The ontology based Fault-Propagation Model is then further processed by using a command line tool that was implemented based on the Jena [APA17] API for parsing RDF and OWL ontologies. The output is an XML file that contains only the information required for run-time reasoning. The XML file is given to an central FDH Configuration Server that interacts with the FPMR repositories of the FDH nodes in the network and supplies them (over a TCP connection) with the latest version of the Fault-Propagation Model. The FPMR repositories in the network devices parse the conveyed XML file and store the Fault-Propagation Model in a form that enables fast real-time Fault-Isolation. Similarly, one can distribute FRF configurations, as well as any other information required for the proper functioning of the FDH components.

The last aspect that needs addressing is the *Network Monitoring System* to use, in order to enable the network operations personnel monitoring the FDH self-healing activities. Therefore, a number of freely available Network Monitoring Systems were evaluated, such as Nagios [Bar08], OpenNMS [OPE17c], and perfSONAR [PER17]. The choice was made to deploy Nagios due to its popularity in the network community and its plug-in based extensibility, which allowed to implement *Perl* and *C* based Nagios plug-ins interacting with the FDH components on a host that is meant to receive each alarm/incident/event occurring in the FDH managed system (the alarms/incident/events are received over the FDH EvDissAgent overlay). To emphasize: this node consolidates all the emerging information regarding FDH self-healing in the network. The Network Monitoring System plug-ins are periodically started by the Nagios daemon and check for conditions which must be escalated based on the criteria defined in section 6.4. Given such a condition, a notification is alerted to the network administrator over the Nagios web interface. That way, the requirements and criteria from section 6.4 can be easily realized. Furthermore, the above construction enables the continuous monitoring of the situation in the network (with respect to faults/errors/failures/alarms) and provides the means to observe Control Loop History Logs and monitor the operation of the FDH agents. Further information on these aspects is provided in chapter 9 on the processes and models regarding the FDH interactions to/from the network operations personnel.

Chapter 11

Testbed Environment for Evaluating the Framework

This section describes the testbed environment that was used for the evaluation of the concepts proposed in this thesis. For this purpose, the FDH prototype - implemented in the course of this work - was deployed in the testbed presented here, and various tests and experiments were conducted.

The environment that was used for the evaluation of the FDH framework is illustrated in Figure 11.1. This testbed was developed at the Fraunhofer FOKUS institute in the course of the EFIPSANS project [efi16a]. It consists of virtual machines (those in grey boxes) being hosted within a VMware ESXi [Hal11] server environment. In addition, a number of physical machines were integrated (those out of the grey boxes), in order to explore different situations based on real (and not only emulated) hardware. Furthermore, in order to concentrate on the IP layer, different types of lower layer access network technologies and devices (e.g. DSL routers, aggregation routers, EPON switches ...) are omitted and only the corresponding IP links put in place for the FDH experimentation. Moreover, IPv6 was used as a network layer protocol. This choice was made for a number of reasons:

1. the increasing importance of IPv6 as a network layer protocol for Future Internet,
2. a number of exciting new features and automations coming with IPv6 and providing space for experimenting with FDH in respect to those features (e.g. the IPv6 Black Hole scenario), and
3. the vision and intensive EFIPSANS effort that was conducted towards implementing (autonomic) Network Management automations on top of IPv6.

Furthermore, a note with respect to the link layer protocols in the Figure 11.1 testbed: The green links on Figure 11.1 denote Gigabit Ethernet links, whilst the rest are Fast Ethernet links, i.e. R1↔R4, R9↔R10, and R10↔R11.

In parallel to the IPv6 network running among the nodes illustrated on Figure 11.1, an IPv4 management network was used to access the machines and manage various aspects of each node (e.g. addressing and routing). In addition, this IPv4 management network, and the belonging IPv4 management interface of each machine, was further used for (automatically) deploying the FDH prototype in the network and running experiments on top of the IPv6 infrastructure.

Table 11.1: Testbed Hardware Parameters for Testbed Network (Figure 11.1)

Machine	Hardware Parameters	Network Element
VMware Server 1, 4, 5	DellWorkStation R5400, 4 CPU × 1,944 GHz, 4091,56 MB RAM	R4, R5 R8, Hosts
VMware Server 2, 3	Dell Power Edge SC1435, 8 CPU × 2,493 GHz, 8187,56 MB RAM	R1, R2, R3, R6, Hosts
PC 1	Intel Pentium 4 CPU 3,00 GHz, 992,96 MB RAM	R9
PC 2	Intel Pentium 4 CPU 3,00 GHz, 2003,94 MB RAM	R10
PC 3	Intel Pentium 4 CPU 2,60 GHz, 488,76 MB RAM	R11

The hardware parameters of the testbed are once visualized on Figure 11.1, and in addition summarized in Table 11.1. The hardware that was used is far from the parameters of full scale backbone routers. However, it provides a reasonable possibility to test the FDH concepts in an environment based on soft routers (Linux Ubuntu/Debian) and open source libraries and components. In this line of thought, it should be mentioned that soft routers are increasingly gaining popularity within the network community, especially when it comes to Local Area Networks and academic environments. Thereby, various commercial products are available on the market, e.g. firewalls or Intrusion Detection Systems, which are based on soft routers using open source operating systems, such as Linux, FreeBSD, NetBSD, or OpenBSD. These facts emphasize the relevance of the used network environment/testbed despite the fact that it lacks real backbone routers, e.g. Cisco or Huawei based devices.

Having elaborated on the relevance of the network environment, the different routing platforms used on the Figure 11.1 network are presented. As previously mentioned, Linux Ubuntu/Debian was used as an operating system for all the nodes. On top of this, the routers in Figure 11.1 were running different routing daemons depending on the requirements of the different EFIPSANS scenarios [KBA+10]. In particular, the Quagga [QUA17] and XORP (eXtensible Open Router Platform) [XOR17] were used for realizing the IPv6 routing functionality in the testbed. Thereby, both platforms were set to execute the OSPFv3 routing protocol, which is the IPv6 ready OSPF version. XORP and Quagga are both platforms that are currently maturing in terms of IPv6 routing support, by providing protocols such as OSPFv3, BGPv4, and RIP-ng. However, both platforms were experienced to have some bugs, e.g. with respect to OSPFv3 areas for Quagga in v0.99, which led to the need to switch between them depending on the requirements of the particular EFIPSANS scenario. For the purpose, of the trials conducted within this thesis, both platforms were sufficiently good. In the following sections, the experiments are described, which were conducted based on the prototype and the introduced testbed.

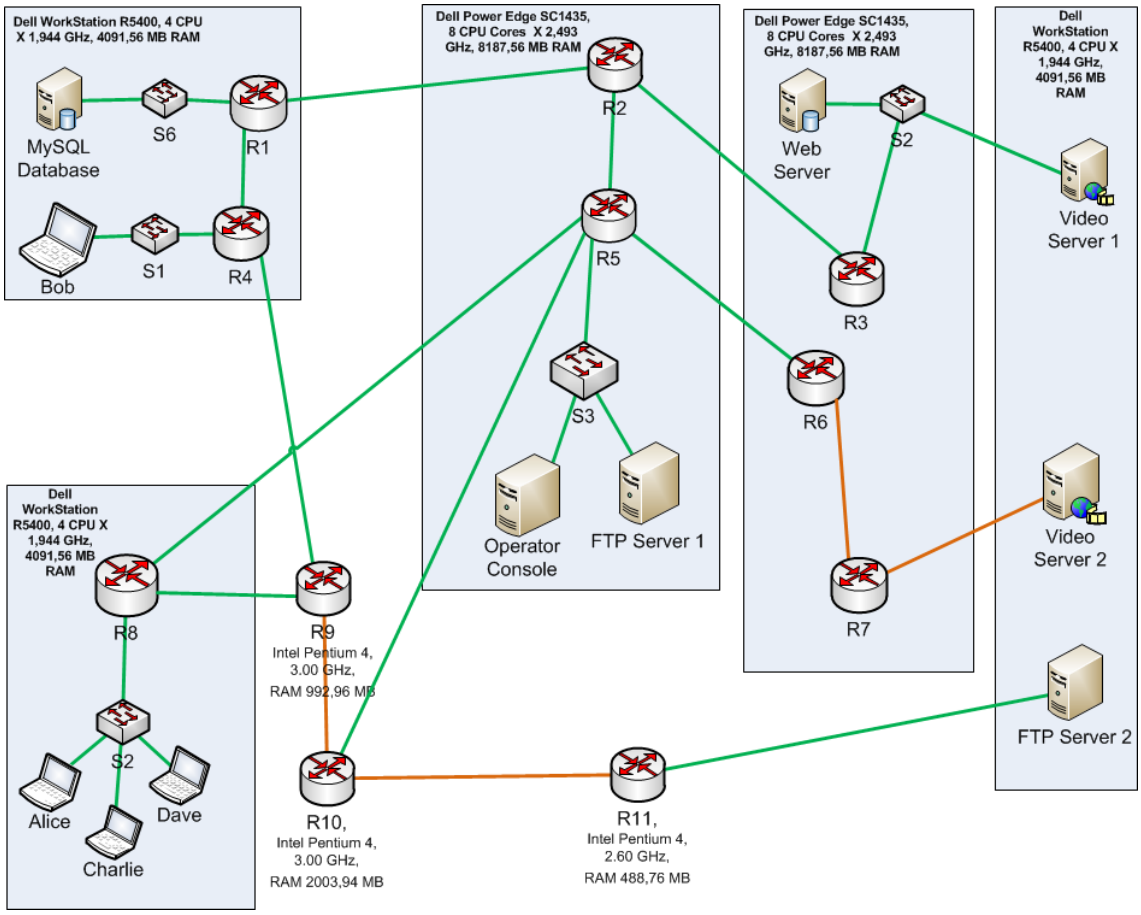

Figure 11.1: Testbed Environment for the Evaluation of the FDH Framework

Chapter 12

Overall Scalability and Overhead Analysis

This chapter presents the scalability and overhead analysis of the proposed FDH framework, thereby addressing **Req 24** and **Req 25** from section 5.1.4. Initially, the processes and overall collaborative behavior of the distributed FDH instances are theoretically analyzed and generally mapped to the operational patterns of established and widely deployed routing solutions. That way, it can be shown that the FDH self-healing control loops behave similarly as currently utilized networking technologies that have proven to scale reasonably in production environments. Secondly, the different modules of the prototype implementation are analyzed by pushing various operational parameters to the extreme and measuring the performance, overhead and scalability characteristics of the single processes. In addition, the overall distributed control loop is evaluated regarding its performance, overhead and scalability in the testbed on Figure 11.1.

12.1 Theoretical Analysis

As pointed out a number of times, the key and most intensive FDH self-healing control loop - realized by the FaultManAgent - involves the processes of Fault-Detection, Alarm/Incident-Dissemination, Fault-Isolation and eventually Fault-Removal. In the scope of the theoretical analysis performed in this section, the above distributed self-healing control loop is mapped to the OSPF routing control loop. That way, one can observe that in general the self-healing control loop introduces a structure which is similar to control loops, which are already widely deployed in production networks. Hence, the general structure of the self-healing control loop would not be expected to overstrain the FDH managed networks/systems and lead to unexpected anomalies. In addition, the key processes of the main FDH control loop are theoretically analyzed w.r.t to expected scalability, performance and overhead in the context of the key FDH self-healing control loop realized by the FaultManAgent.

12.1.1 General Approach

The general approach towards showing that the primary mechanisms, proposed in chapter 8 and chapter 8, as well as the overall self-healing control loop have promising scalability properties, involves two aspects:

1. Try to map and identify similarities (as far as possible) between the FaultManAgent agent control loop structures and the OSPF control loop, which is a successful and well-proven routing solution available in practice, and

2. The complexity analysis and performance evaluations provided along with the proposed mechanisms (chapter 8 and chapter 8 are discussed in the context of the FaultManAgent agent control loop.

Coming back to the first point from above, the OSPF protocol is widely used in modern networks and provides a number of self-X features such as self-healing, self-description and self-adaptation, in order to automatically establish an optimal and reliable IP topology [RNCS09]. Focusing on the general structure of the OSPF control loop, one can observe that it is pretty similar to the FDH FaultManAgent control loop. On the one hand, OSPF uses the Hello Protocol for detecting node and link failures. On the other hand, the FDH employs diverse monitoring techniques for detecting incidents in the network. While OSPF disseminates LSAs (Link State Advertisements) - based on the results from the Hello Protocol [Moy98] - using flooding across the routing area, the FDH FaultManAgent age control loop employs multicast/unicast based flooding to spread information about detected incidents across the network (area).

With the SPF (Shortest Path First) computation, the OSPF control loop analyzes the link state updates, obtained by the Hello Protocol [Moy98] and conveyed to each router by the LSA flooding mechanism. Similarly, the Fault-Isolation in each FDH node, following the Fault-Detection and Incident Dissemination processes, analyzes the symptoms of an erroneous state detected across the network area.

The last step in the OSPF control loop is the FIB (Forwarding Information Base) update in every router, and the corresponding phase of the FaultManAgent control loop is the execution of a Fault-Removal action.

However, the FDH FaultManAgent control loop is more complex than the OSPF one, since it may require the synchronization (with the goal of optimizing the overall set of FDH actions) of the Fault-Removal action (potentially also of any Fault-Mitigation action), and involves the assessment of the executed Fault-Removal task, and in the worst-case the escalation to a network level FaultManAgent agent, and respectively to the network operation personnel.

Within the following sections, the different mechanisms are analyzed which were proposed for the primary processes of the most intensive FDH control loop - the FaultManAgent control loop. These mechanisms are also compared to the corresponding mechanisms of the OSPF control loop.

12.1.2 Fault-Detection

The process of Fault-Detection can be realized by monitoring functions which analyze traffic, monitor the logs of the network nodes, interact with SNMP MIBs, or actively inject traffic into the network in order to check its health and state. For instance, the Hello Protocol [Moy98] employed for monitoring adjacent nodes and links within OSPF is an active type of probing technique. It is obvious that all those techniques will require additional processing, i.e. CPU cycles, and even bandwidth. In this line of thought, the process of Fault-Detection will be as scalable as the employed monitoring techniques. Certainly, a trade of is required between the achieved results of the FDH based self-healing and the overhead for detecting incidents.

Another "monitoring"("information sharing" technique for Fault-Detection that has been men-

tioned in section 5.1.3 is based on the requirement (**Req 12**) that functional entities - protocols, services and applications, should collaboratively share information regarding all incidents they experience. This is a reactive mechanism (say interruption, trap, or exception like) that will not introduce additional overhead for Fault-Detection since there would be no special entities looking for incidents, but the functional entities themselves will report the corresponding incident descriptions as soon as faults get activated. However, this type of requirement is difficult to realize in an evolutionary type of approach to future networking, where protocols/application/services are already in place and have been developed without the awareness for FDH based self-healing.

The Fault-Detection process stores the description of the detected incidents into the FDH node faults/errors/failures/ alarms repositories within a network node. A critical issue with respect to scalability is related to the amount of information that might be stored in those repositories. A step towards remediating this issue is the fact that specific repositories, meant to efficiently handle incident descriptions were implemented in order to reduce the amount of administrative information (such as primary and foreign keys) which would be required in case of employing database products. Moreover, as defined as an escalation criterion in chapter 6.4, a pre-defined threshold should be used for the maximum number of incidents - reported to the FDH repositories - that determines the point, at which the current network state should be escalated to the network operation personnel. This means that an overload of the FDH repositories is indicative for a situation that cannot be handled even by the FDH control loops, and hence they will stop being active. This strategy is expected to put limits and control the FDH behaviors with respect to the amount of data stored in the local node repositories.

12.1.3 Alarm and Incident Sharing

The next scalability issue in the processes of self-healing is related to the alarm/incident information dissemination. This process is vital for the FDH agents throughout a network since it enables them to have a consistent network-wide view of the situation with respect to faults/errors/failures/alarms. Drawing experiences from the OSPF control loop [STP+15], which is available in practice, provides hints on how the problem of scalability with respect to the dissemination of control loop relevant information can be tackled. OSPF disseminates information required for Shortest Path Computations in the form of LSAs (Link State Advertisements) by employing a reliable flooding scheme. This flooding scheme guarantees the reliable exchange of LSA information between two neighbors. It is obvious that the scope of the flooding must be restricted such that the produced overhead does not impact the network negatively. This is one of reasons why the operator's network is separated in different OSPF areas of a size that can be flooded with LSAs within a reasonable time [STP+15]. Correspondingly, the operation region of OSPF is restricted to such a single area. So-called area border routers provide links to other areas (to the backbone area) within the domain. Discussions within the EFIPSANS project [Tch10] suggest that a typical size of such an area is around 50 routers. Hence, when it comes to larger (e.g. telecom type of) networks the alarm/incident dissemination approach should prove scalable in such bounds, i.e. in networks/areas of around 50 routers.

The techniques for alarm/incident dissemination that were presented in section 7.2 can be considered scalable or not breaking the traditional bounds of an IP network for the following reasons:

1. The incident information is disseminated (flooded or gossiped) only to those nodes in the network area where the entities that reside are associated with the incident information to disseminate,

2. In the worst-case of an area of N fully connected routers, OSPF would require $O(N^2)$ packets to deliver an LSA across the area.

The same magnitude of packets is also required by the flooding for alarms and incidents as described in section 7.2.

An interesting aspect is given by the LSA dissemination limitations in the scope of OSPF. Experiences with OSPF in the testbeds of large scale European [STP+15] and national projects involving multiple testbed partners (including Fraunhofer FOKUS) have shown that there are considerable delays in the recovery of an IPv6 topology after multiple link failures. This delay can be partially attributed to the LSA flooding procedure that relies on the direct updates of link state information between neighboring nodes, such that the link state information is gradually distributed across the routing area. In order to accelerate this process, the FDH alarm/incident dissemination sends the corresponding messages directly to IP addresses, which may stand for single nodes or for whole multicast groups of nodes. That way, a step is made towards accelerating the flooding procedure for the sake of rapid self-healing reactions to alarms and incidents in an FDH managed network.

12.1.4 Fault-Isolation

Once the appropriate FDH nodes in the network have been updated regarding relevant alarms and incidents, every FDH instance, in the corresponding devices, starts performing the same Fault-Isolation process/algorithm. Indeed, Fault-Isolation gets automatically initiated within the FDH nodes in the cases identified in section 6.2.1.3:

1. A pre-defined number of unanalyzed incidents and alarms have been reported to the local node repositories
2. An incident with a high severity level (according to a pre-defined threshold) has been reported
3. A pre-defined time-slice has expired since the last time Fault-Isolation was initiated

The process of Fault-Isolation is extremely critical with respect to scalability for two reasons:

1. Computing time required for automated Fault-Isolation
2. Size and memory requirements of the Fault-Propagation Model to analyze

Next, these two aspects are analyzed in turn.

12.1.4.1 Time Complexity

The current section elaborates on the time complexity aspects of the Fault-Isolation procedure that is realized within the FDH node. The potential usage of traditional Bayesian Networks for the purpose of Fault-Isolation is looked at and contrasted to the Markov Chain based Fault-Isolation technique, which was developed in the scope of this thesis. In addition, the time complexity aspects of the Markov Chain based algorithm (section 7.3.2) are theoretically analyzed.

With respect to the application of Bayesian Networks for Fault-Isolation, the reasoning algorithm is in general NP-hard but there exist linear algorithms that work on Bayesian Networks with special structures, such as for example for trees [WZC08].

The NP-hardness of the Fault-Isolation based on general Bayesian Networks was the reason to develop the Markov Chain based reasoning framework described in section 7.3.2. For this Markov Chain based framework the following time complexity was derived - $O(|F||E||I|)$ with $|F|$ denoting the number of faults in the FPM[1], $|E|$ standing for the total amount of events in the FPM, and $|I|$ the number of detected incidents to correlate. This is a polynomial time complexity similar to those of many graph algorithms and is not expected to produce any unacceptable overhead. Moreover, the parameter $|I|$ is explicitly pre-configured by the human experts tweaking the FDH processes. Thus, by choosing small values for $|I|$ (if reasonable) the time complexity for the Markov Chain based reasoning framework could be further improved.

The SPF (Shortest Path First) computation within the Dijkstra algorithm [CLRS01] employed widely in OSPF has a quadratic time complexity with respect to the number of routers. The time complexity of the Markov Chain based approach to Fault-Isolation, proposed in this thesis, may be worse or better (w.r.t. the alarms/incidents captured in the FPM) depending on the structure of the FPM (e.g. number of faults) and the values chosen for the $|I|$ parameter. For example, in case of 100 incidents, i.e. $|E|=100$, and $|I| = 10$, but only 5 root causes/ faults, i.e. $|F|=5$, then $|F||E||I|=5000$ and one can expect around 5000 required operations for Fault- Isolation, which is less than $|E|^2=10000$ required in the case of pure quadratic complexity (as in the case of the Dijkstra SPF computation[2]). Hence, one can see that at least the theoretical time complexity of the proposed Markov Chain based approach is in the dimensions of the SPF computations for OSPF.

12.1.4.2 Space Complexity

As previously mentioned, the other critical issue is related to the size of the Fault-Propagation Model, a copy of which is stored in each device implementing FDH.

In case of a traditional Bayesian Network, a number of conditional probability matrices must be stored for each single event in the FPM. Each matrix is dimensioned according to the number of direct graph neighbors of an incident event. Obviously, a Bayesian Network can easily overload the memory of a router with the growing number of faults/errors/failures/alarms whose relations must be captured. This was another reason that led to the research on the Markov Chain based approach presented in section 7.3.2.

The space complexity for the FPM within the Markov Chain based framework is $O(|F||E\backslash F|+|E|^2+|E|)$ or simplified $\boldsymbol{O(2|E|^2+|E|)}$ if one takes $|E|$ as an upper bound for $|F|$ and $|E\backslash F|$. In the following analysis it is assumed that each incident is referred as a two bytes integer number. Presuming that 10MB are allocated in each network node/device for storing the FPM, the following quadratic equation emerges for the number of events and their relations that can be captured:

$$2 \mid E \mid^2 + \mid E \mid = (10 \times 1024 \times 1024)/2 \tag{12.1}$$

The positive solution of (12.1) is 1618, which means that with 10MB the relations among 1618

[1] FPM stands for Fault-Propagation Model.

[2] The Dijkstra SPF computation is quadratic w.r.t to the vertices of the graph. However, the current comparison shows that the FDH Fault-Isolation, based on the Markov Chain technique proposed in this thesis, runs in the magnitude of the SPF computation within OSPF.

faults/errors/failures/alarms across the corresponding network area/scope can be captured. For example, given a network area of 50 routers, it is possible to have $\lfloor 1618/50 \rfloor = 32$ incident events related to each router. In case 100MB are allocated for the FPM in each device, the relations among 5119 faults/errors/failures/alarms across the corresponding network area can be captured. This results in $\lfloor 5119/50 \rfloor = 102$ potential alarm/incident events per node. These calculations are meant to give the reader a feeling about the amount of data required for the process of Fault-Isolation when based on the Markov Chain based framework developed in the scope of this thesis.

12.1.5 Fault-Removal and Fault-Mitigation

After the right root cause has been identified, the process of Fault-Removal consists of a direct reaction to the isolated fault(s) on the corresponding nodes/devices. Similarly, the Fault-Mitigation Functions of the SurvAgent are activated in order to issue an immediate reaction to alarms or incidents, which indicate anomalies and potential problems within the networked system in question. The overhead produced by the Fault-Removal/Mitigation reaction depends on the scripting automation or CLI triggered to resolve the problem.

In case the tentative reaction requires synchronization, the SelfOptAgent should be referred. The SelfOptAgent and the belonging mathematical framework, which was developed in the scope of this thesis, have been analyzed in terms of response times for different numbers of parallel synchronization requests. The results are presented in the coming section 12.2.5. The upper bounds of the response times do not exceed 0.12 sec, which is presumed acceptable for the purposes of FDH in the long-term operation of the network.

The SurvAgent control loop executing fast Fault-Mitigation functions is also expected to refer to the SelfOptAgent. In case the Fault-Mitigation action is urgently required, then the special functionality of the SelfOptAgent that triggers immediate synchronization (see message formats and interface descriptions in section 10.5.2) should be employed in order to speed up the Fault-Mitigation reaction. Since the FMF are similarly based on script automations and CLI calls, the considerations (beginning of this paragraph) made regarding Fault-Removal are equally valid here.

12.2 Empirical Analysis

In this section, the analysis of the architecture continues by presenting the results from various simulations (e.g. OMNET++ simulations) and experiments conducted in the testbed illustrated in Figure 11.1. The FDH self-healing starts with the task of Fault-Detection, which heavily depends on the deployed monitoring tools and on the overhead they produce within a device. The tasks which are clearly to be classified as FDH mechanisms are Incident- Dissemination, Fault-Isolation, Action Synchronization performed by the SelfOptAgent, Fault-Removal/Mitigation and other tasks which depend on the proprietary policy handler that was implemented for the purposes of FDH. In the following, each of these tasks is analyzed in turn.

12.2.1 Monitoring and Fault-Detection Processes

Besides the mere execution/invocation of monitoring tools, the Fault-Detection task encompasses various processes which are implemented by the MonAgent. These include the processes of

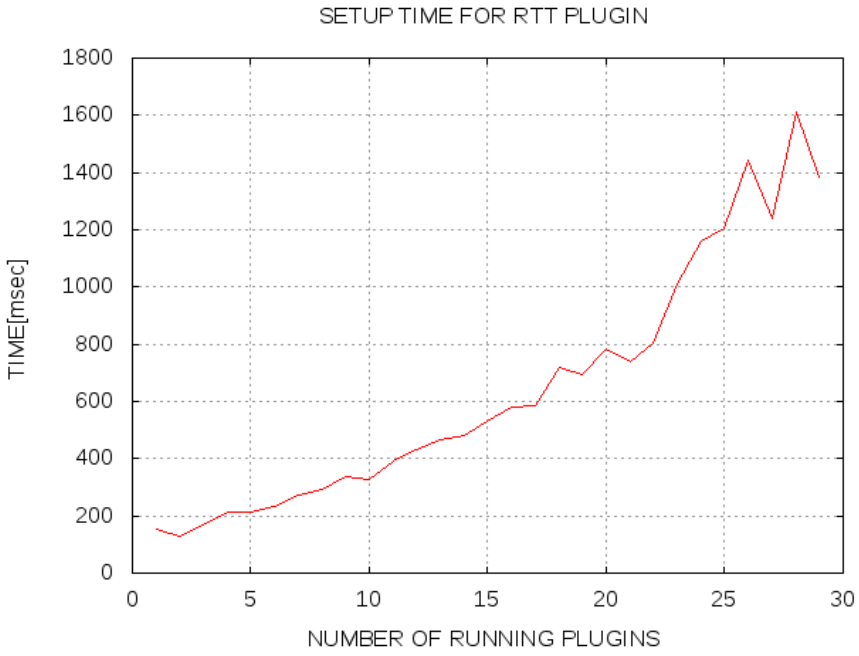

Figure 12.1: Overhead Measurements: Setup Time for an RTT Plug-in

1. setting up a monitoring job based on a request submitted by another FDH agent and of
2. periodically executing monitoring tools within the belonging monitoring plug-ins, which automate the interplay of single monitoring tools.

Based on the prototype described in section 10.2, it is an interesting task to examine the above processes with respect to scalability and resource consumption thereby obtaining a feeling regarding the belonging operational parameters.

The experiments presented here were conducted on a Linux Ubuntu desktop machine with the following parameters: Kernel Linux 2.6.32-36-generic-pae, Intel(R) Core (TM), 2 × "2.80GHz Duo CPUs", 3.9 GB RAM. Following aspects were evaluated in particular:

1. required time and memory consumption for setting up a number of monitoring jobs and
2. operational memory consumption while running a maximum number of plug-ins.

At this point, it must be said that the maximum number of plug-ins that could be run simultaneously and in a single FDH MonAgent was around 30. Trying to run larger numbers of simultaneous plug-ins resulted quite often in "too many open files" exceptions. Eventhough an effort was undertaken to close all streams properly, obviously there is more need for optimization and improvement of the Java code of the MonAgent.

Figure 12.1 illustrates the time required for instrumenting up to 29 RTT monitoring plug-ins, which may be use to detect connectivity and large delay issues within a network. Similarly, Figure 12.2 shows the time required for instrumenting up to 29 QoS plug-ins. The QoS monitoring plug-in instruments the *iperf* [IPE17] tool, in order to measure key QoS related KPIs such as jitter and

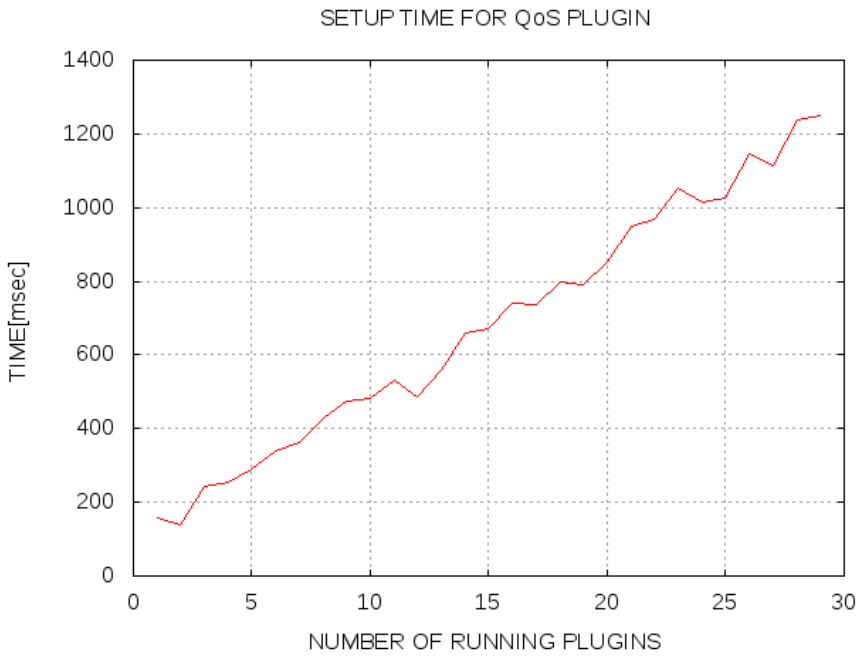

Figure 12.2: Overhead Measurements: Setup Time for a QoS Plug-in

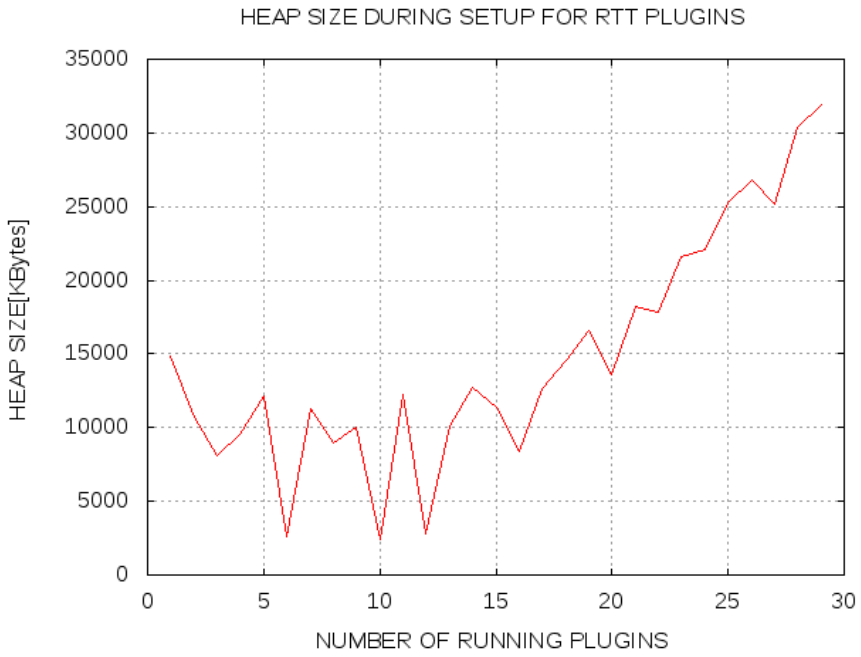

Figure 12.3: Heap Size during Setup for RTT Plug-ins

"out of order packets". In both cases the instrumentation of the monitoring job is achieved pretty fast, and the monitoring service is ready to provide the required monitoring information within less than two seconds. This implies that the current FDH MonAgent prototype can potentially be used for real time tasks since it can quickly prepare to provide required monitoring data.

Figure 12.4: Heap Size during Setup for QoS Plug-ins

Figure 12.3 and Figure 12.4 illustrate the heap memory consumption during the above analyzed monitoring job setup processes. One can clearly observe the operational cycles of the Java garbage collector visible through the sudden drops in the amount of consumed memory. Besides, one can see that memory consumption for the setup operation goes in the magnitude of dozens of megabytes. This result can possibly be improved within a C/C++ implementation given that the MonAgent of the FDH needs to operate in environments where memory is an extremely scarce resource. Finally, Figure 12.5 visualizes the heap size during the operation of 30 simultaneously running plug-ins. Half of the plug-ins were RTT plug-ins, and the other half QoS plug-ins. The Java garbage collector is again clearly visible with its operating cycles.

In addition, the previous findings regarding memory consumption are again confirmed since a magnitude of dozens of megabytes is continuously observed. Thus, it can be recognized that the current MonAgent prototype is only limited applicable to small devices, e.g. sensors etc. However, it has quite suitable characteristics for operating within a traditional host machine or in the control plane of a router. In such deployment scenarios, FDH MonAgents can be either deployed on each involved node, or just on particular dedicated machines/routers, in a way that they can deliver monitoring information and realize Fault-Detection tasks for the purposes of the distributed FDH based self-healing.

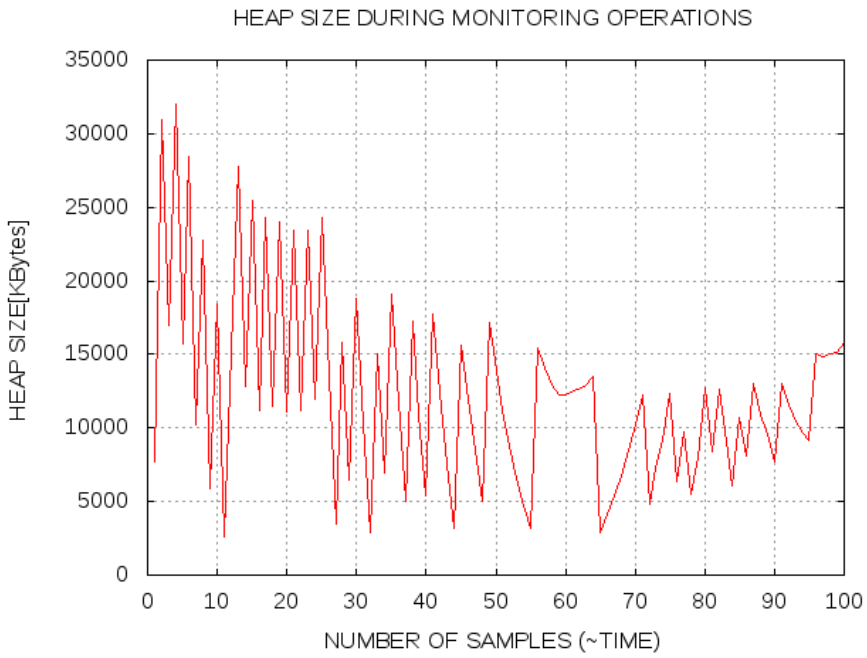

Figure 12.5: Heap Size during Monitoring Operations

12.2.2 Incident Sharing

As previously mentioned, a canonical type of flooding on top of IP was implemented as a first solution for the FDH prototype. In the course of the procedure, the symptom detecting node pushes initially an alarm/incident/event message to a list of IP(v6) addresses, including multicast addresses as obtained within MLD for IPv6 or IGMP for IPv4. As a result, every receiver node forwards the alarm/incident/event message to all IPv6 addresses, if it sees the message content for the first time, or dismisses it in case the message is locally known. This procedure worked sufficiently during the trials conducted in the testbed illustrated in Figure 11.1.

For the purpose of evaluating the scalability of the Incident-Dissemination an OMNET++ [VH08] simulation was put in place. In general, the same simulation setup was used which was already presented in section 7.2.3, in the course of evaluating the quality of the dissemination procedures. Thereby, networks of different sizes (10,50, 100, 200, ...,1000) were simulated. These were random networks based on the Waxman model [Nal05] for Internet topologies. For the generation of these networks, the BRITE tool [BRI16] [MLMB01] for random Internet topology generation was utilized. Some of the employed topologies are displayed in Figure 7.6 and in Figure 7.7 in section 7.2.3.

The conducted experiments were used to compare the implemented flooding procedure with the repetitive retransmission of datagram incident messages issued by the node where Fault-Detection has taken place. Furthermore, for the time of each trial, every "machine" in the topology was generating background traffic towards random nodes in the simulated network. In the course of this, a packet of size 1024 bytes was injected into the network every 100 microseconds by each simulated node. As previously mentioned, this resulted in at least a dozen of Giga-Bytes of background traffic for each simulated network.

The focus in this section is set on the aspect of the time required for all involved network nodes, in

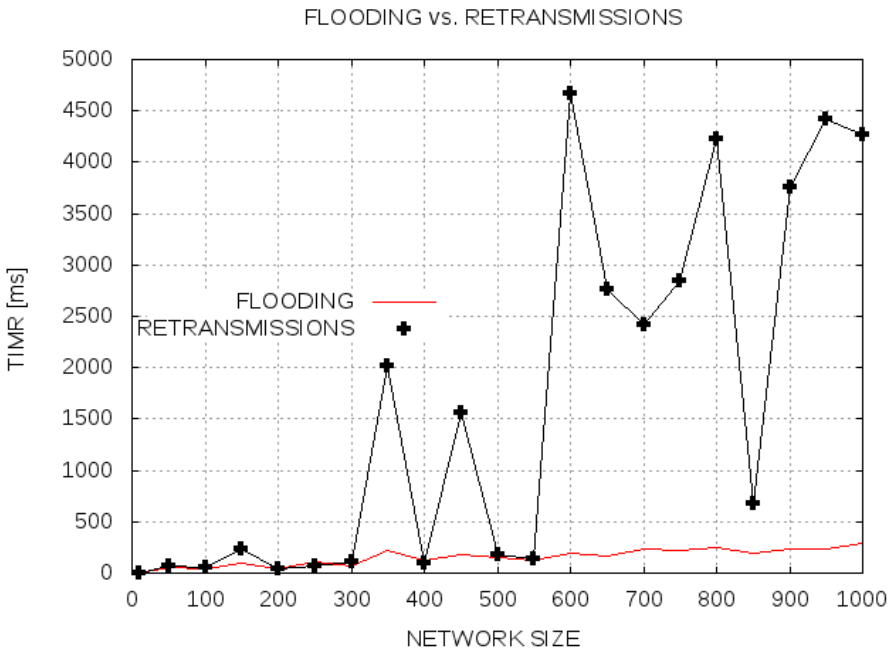

Figure 12.6: Incident-Dissemination Convergence Times

order to have received the incident information in question. The simulation results regarding the convergence times in random networks (based on the Waxman model [Nal05]) of different sizes are illustrated in Figure 12.6.

Figure 12.6 shows that the flooding procedure - implemented in the current prototype - achieves pretty low convergence times, while the datagram retransmission fluctuates in terms of required convergence time thereby generally performing worse than the implemented canonical flooding.

The evaluation provided in the previous paragraph should be seen as a result of a first proof of concept attempt to evaluate the Incident Sharing process. However, given the recent advances in the area of information dissemination and message routing, this topic offers a large potential for applying diverse techniques and investigating their scalability and effectiveness in large networks.

12.2.3 Fault-Isolation

The Fault-Isolation component of FDH is implemented based on the Markov Chain based algorithm presented in section 7.3.2. This algorithm operates on a Fault-Propagation model that, similarly as for Bayesian Networks, is represented as a directed acyclic graph of events. Thereby, the Bayesian Network reasoning procedure was reformulated as to infer the most likely sequence of events based on the detected symptoms, as opposed to Bayesian Networks that infer the probability of each single event to have occurred based on the evidence, i.e. the detected symptoms (incidents and alarms). This allows for implementing a highly efficient Fault-Isolation procedure which scales with respect to required memory and time for completing the reasoning. The key results are illustrated in Figure 12.7 and clearly confirm the scalable properties of the Fault-Isolation mechanism implemented within the FDH prototype.

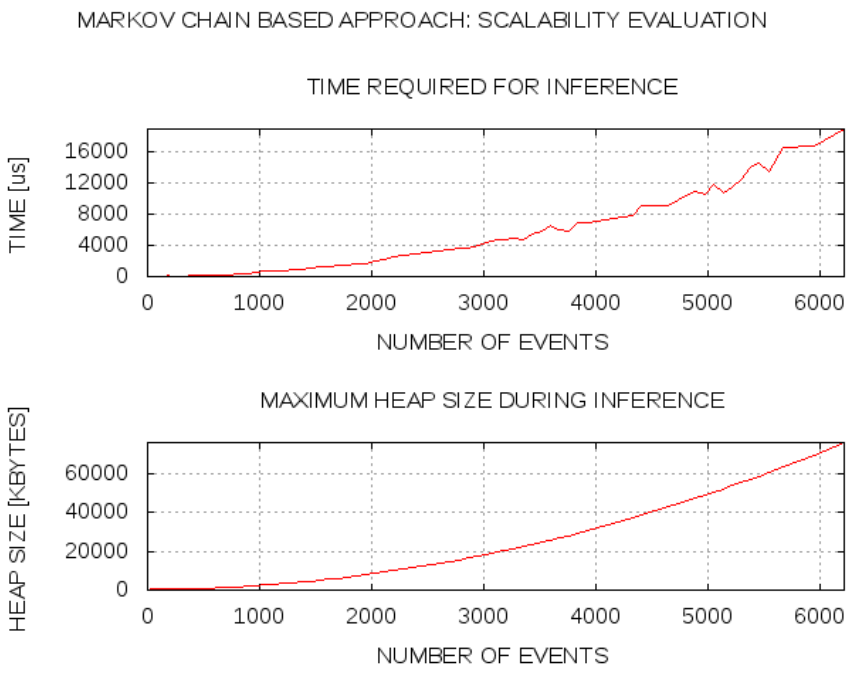

Figure 12.7: Scalability Evaluation of the Fault-Isolation Mechanism

12.2.4 Policy Handling

The FDH framework defines a number of components that were realized based on a proprietary C++ implementation of a policy handler that can operate on XML input artifacts similar to those described in chapter 10. These components are the Fault-Isolation Assessment Functions, the Fault-Removal Functions, the Fault-Mitigation Functions, the Fault-Removal Assessment Functions, and the Asserted Incidents Assessment Functions.

In order to get some indicative measurements for the performance of the policy handler, policy sets with various sizes were randomly generated. The time required for the evaluation of these policy sets is plotted against the policy set size in Figure 12.8. It can be observed that the policy evaluation time is in the magnitude of microseconds and hence the only potential source of performance problems remains the execution of the CLI required as a result of the policy evaluation.

12.2.5 Action Synchronization within the SelfOptAgent

The SelfOptAgent implements an action synchronization procedure, which was realized based on the techniques presented in section 8.2. The idea is to represent the actions with their potential influence on diverse KPIs in a way that techniques from the area of mathematical optimization are applicable (more details are given in section 8.2). As previously mentioned, an open source solver package [COI17] was used for implementing the action synchronization procedure.

The results of the empirical scalability and overhead evaluations - based on directly solving the binary Action Synchronization Problem - are presented in Figure 12.9. In order to evaluate the SelftOptAgent and the belonging action synchronization procedure, 1000 instances for each model size in question were simulated. Thereby, each instance had an equal number of potential actions and KPIs that are influenced by these actions. In addition, two threads were used, which simulated

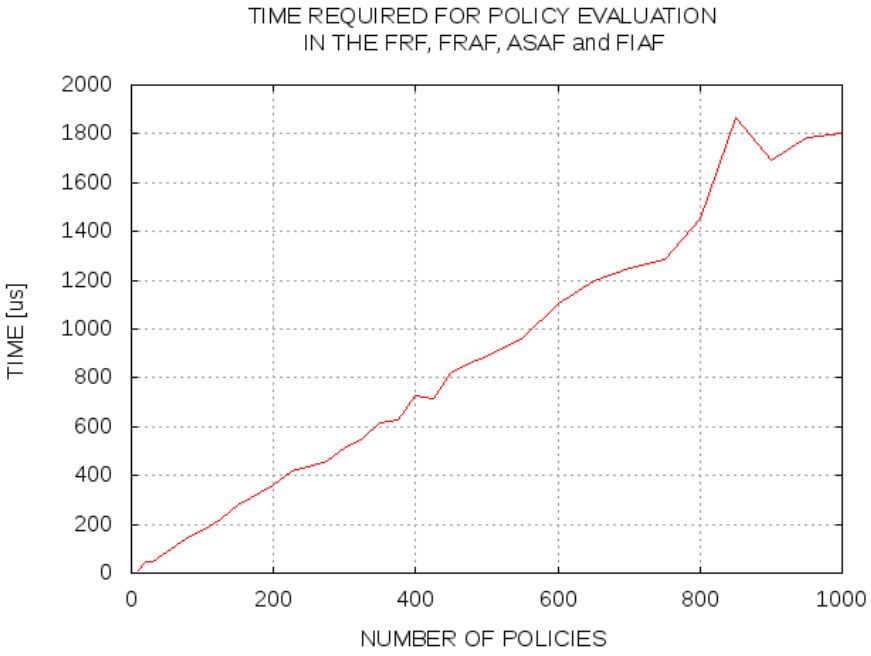

Figure 12.8: Scalability of the Policy Handler based Components

the simultaneous issuing of requests (as expected from the FDH agents) for all the instances of each model size. The maximum of the response times and the required heap memory for each model size in the course of these experiments are presented in Figure 12.9. These measurements indicate reasonable scalability properties and moderate memory requirements for the SelfOptAgent part of the FDH prototype.

Furthermore, it is of special interest to observe the empirical scalability characteristics of RASP (Relaxed ASP) and compare those to the characteristics of ASP. Figure 12.10 visualizes the comparison between both approaches with respect to the maximum execution time for the instances of a particular model size. In addition, Figure 12.11 reflects on the memory consumption when solving RASP and correspondingly ASP with a growing model size - similarly the maximum observed heap memory consumption over all instances of a particular size is taken. It can be observed that RASP is clearly superior to ASP with respect to empirical overhead and scalability - both regarding execution time and memory consumption. Hence, a RASP solving SelfOptAgent is clearly the better choice in cases when memory is a scarce resource and fast execution times for the self-optimization task are required.

12.2.6 Scalability and Overhead Evaluation of the overall distributed Control Loop

Next, the presentation proceeds with an overall evaluation of the distributed FDH self-healing control loop. This includes the interplay between the different processes analyzed before, however without pushing every single process to the extreme as done in the previous paragraphs. The experiments were conducted in the testbed described in Figure 11.1.

The scenario for these experiments consisted of the following steps:

1. the process of Fault-Detection was simulated on R4 (a virtual image with *Intel(R) Xeon(R),*

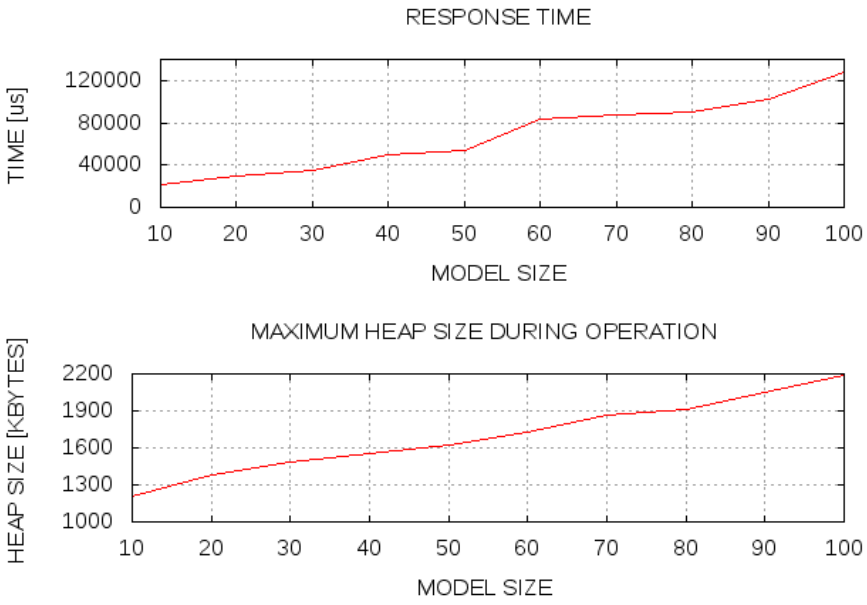

Figure 12.9: Scalability of the SelfOptAgent when solving Binary ASP

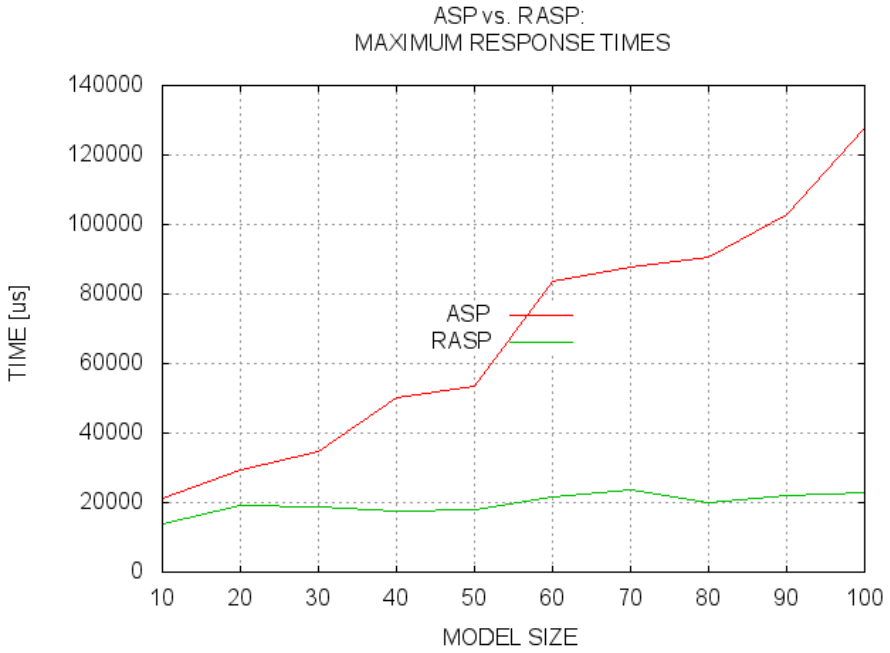

Figure 12.10: Maximum Response Time for each Model Size when solving ASP and when solving RASP

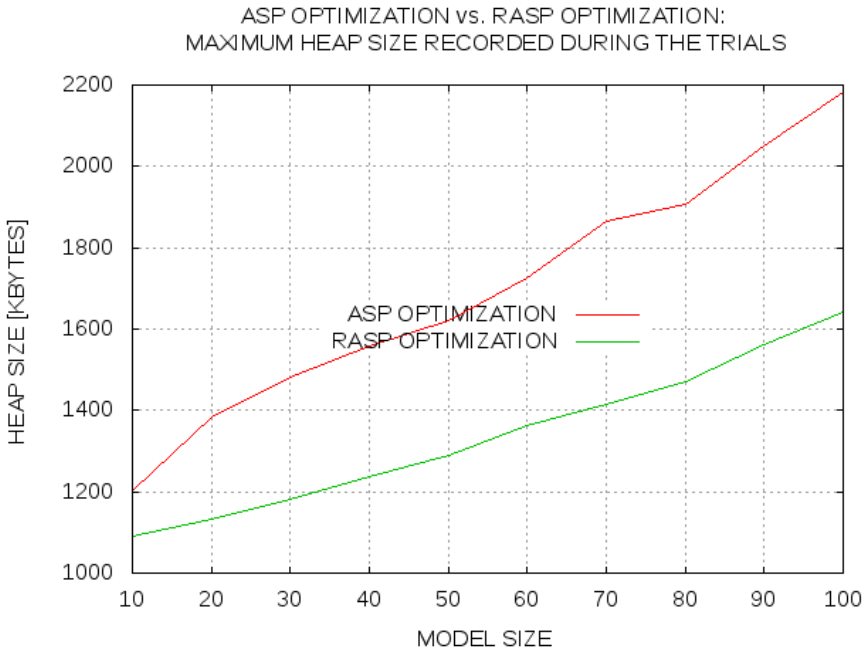

Figure 12.11: Memory Consumption when solving ASP and when solving RASP

"CPU 2.00GHz", 512 MB RAM) by an entity that periodically pushed incident descriptions to the local set of incident repositories,

2. the information was pushed to the local FaultManAgent and forwarded to the EvDissAgent on R4,

3. the EvDissAgent on R4 disseminated the incident descriptions across the whole testbed (including routers and end systems),

4. each router and end system performed Fault-Isolation upon receiving the incident,

5. all devices, except for R3 (virtual image with *Quad-Core AMD Opteron(tm) Processor 2380, 1024 MB RAM*), were configured to back off with respect to Fault-Removal while R3 performed action synchronization over the SelfOptAgent and executed the *touch* [TOU17] Linux command creating a file locally and emulating the process of Fault-Removal,

6. the Fault-Removal Assessment was similarly simulated and followed by the dissemination of a fault recovery message across the network,

7. finally, after the recovery notification was received back on R4, the time was measured between the submission of the incident description to the R4 repositories and the arrival of the recovery message.

The above described procedure was repeated 1000 times in a row in order to gain data regarding the overall execution time of the distributed control loop. This data is plotted in Figure 12.12 and shows that in the current test setup, the execution time for the scenario, simulating the overall distributed FDH self-healing control loop, was in the magnitude of several thousands of microseconds.

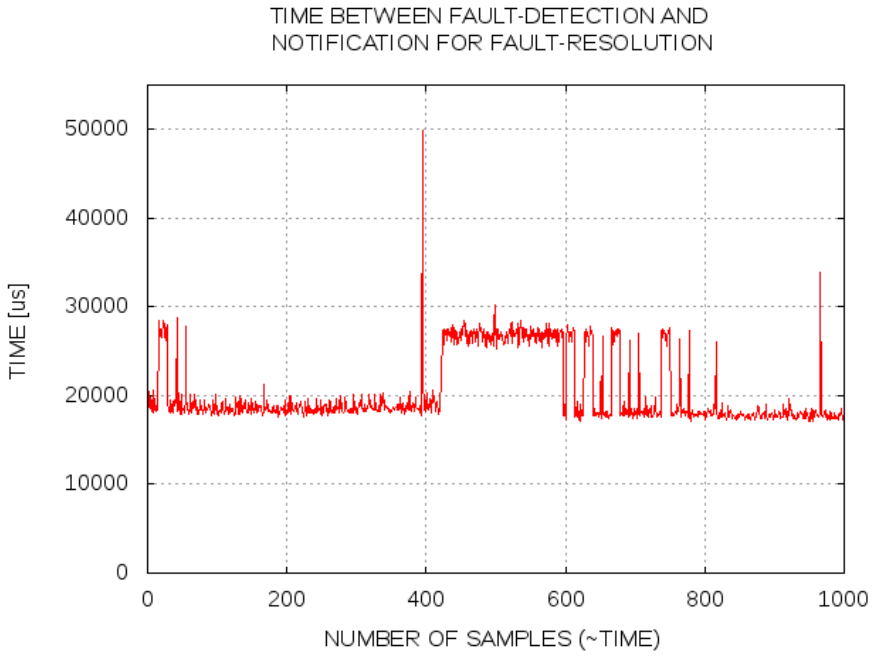

Figure 12.12: Execution Time of the overall AFH Control Loop

In parallel to executing the above procedure and obtaining the data plotted in Figure 12.12, the *top* [TOP17] Linux tool was used in order to get a feeling regarding the CPU utilization on the key routers (R3 and R4) within the experiment. The CPU utilization was measured in intervals of five seconds. Thereby, the maximum CPU utilization of the overall set of FDH components was measured to be at most 0.2% within the intervals. This is a very minimal value which can be explained by the fact that the experiments tested primarily the transfer/communication of alarm/incident messages between the different FDH components and the FDH nodes in the Figure 11.1 network. These processes are not as computational intensive as the Fault-Isolation and Action Synchronization processes, which were evaluated in the previous sections. However, these measurements show that FDH causes minimal CPU utilization - on a *Quad-Core AMD Opteron(tm) Processor 2380 and Intel(R) Xeon(R), "CPU 2.00GHz"* - for the pure passing of alarm/incident messages between the FDH components (as specified in the sequence diagrams in section 6.3).

Finally, additional 50 runs were performed in order to gain experience regarding the FDH memory consumption on the key routers in the scenario, i.e. R3 and R4. The results regarding the heap size memory consumption are plotted in Figure 12.13, and show again some minimal overhead in the magnitude of dozens of Kbytes. These measurements complement the "component by component" evaluation of the previous sections, where each single component was pushed to the extreme, and give an idea of the scalability of the interplay among the overall set of employed mechanisms and algorithms.

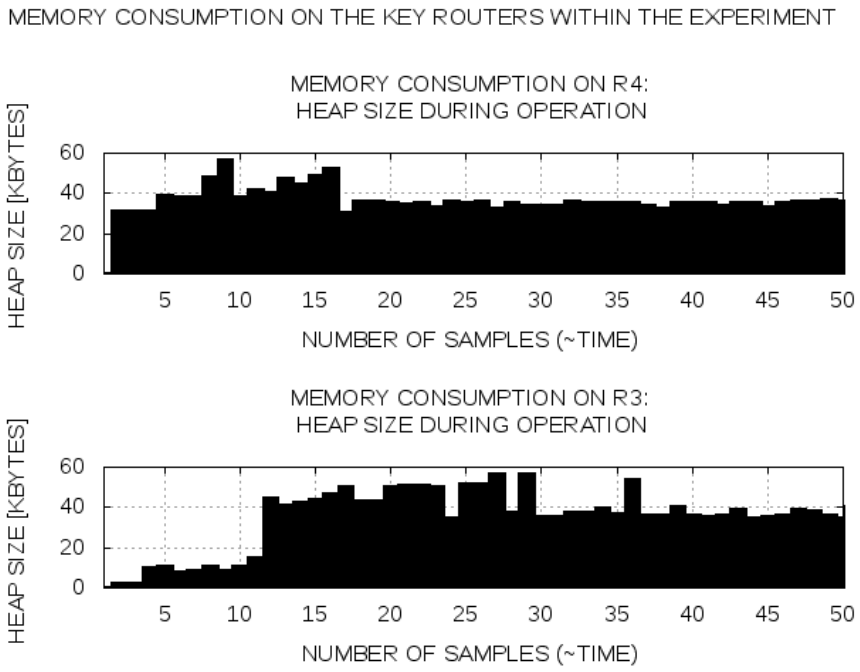

Figure 12.13: Memory Consumption measured during the Execution of the overall FDH Control Loop

Chapter 13

Case Studies

The prototype presented in the previous sections was further used to emulate and test the applicability of the FDH framework on different case studies, which are based on IP networks. The first two case studies deal with networking problems on the level of network, link and physical layers. The following third case study is a theoretical one and relates to an architecture for mobile wide-area surveillance running on top of IP technology. Finally, a combined case study consisting of multiple problems introduced to a larger testbed (refer to Figure 11.1 for the testbed) illustrates how FDH can increase the resilience of an operator's network, even when multiple faults are continuously introduced.

Some of these case studies are based on a survey of relevant networking problems that was conducted in the scope of the large scale EU EFIPSANS project [Tch10]. In addition, a research collaboration between Fraunhofer FOKUS and VTT has influenced the case study related to the mobile surveillance architecture. Finally, the combined case study is a demonstration regarding how FDH can solve complex challenges, which are largely based on the single case studies - including the case studies used for demonstrating aspects like Fault-Detection, Fault-Isolation and Self-Optimization - in a testbed [KBA+10] that was designed for the needs of the large scale EU FP7 EFIPSANS project [efi16a].

13.1 A Black Hole Problem in IPv6 Networks

First, a case study is presented which is based on an IPv6 problem that may arise in the process of IPv4-to-IPv6 transitioning, which is imminent on large scale in the coming years. This networking problem was already described in previous sections and used as basis for the case studies on Monitoring/Fault-Detection (section 7.1.3), Fault-Isolation (section 7.1.3) and Self-Optimization (section 8.2.7). For that reason, only a high level recap of the problem characteristics is given here. After briefly presenting the problem itself, the resolution which was realized and tested in the scope of FDH is elaborated.

The presented type of Black Hole matches to the type of faulty conditions presented in section 5.3.1 and section 5.3.4, i.e. *Faults/Errors/Failures detected at the Network Layer, Link and Physical Layers* and *Human Errors during Network and Service Management Phase*. The classification results from the fact that the presented Black Hole type is due to a firewall misconfiguration which affects the IP layer of an IPv6 network.

13.1.1 Description of PMTU Black Holes in IPv6 Networks

The current case study builds on the PMTU Black Hole problem described in [Lah00]. As previously mentioned, the following paragraph is a short recap of the Black Hole problem, as introduced in previous chapters and used there for demonstrating the FDH processes of Fault-Detection/Monitoring, Fault-Isolation and Self-Optimization.

The term Black Hole stands for a packet forwarding node that discards packets of a flow, without notifying the node from which the traffic originates. Hence, the sender of the packets is disabled in reacting to the faulty condition in the network, which results in a communication that stalls/freezes on the end user's system. In [Lah00], a version of the Black Hole problem is presented, which has the potential to occur in IPv6 networks that have adopted an IPv4 firewall configuration (e.g. due to IPv4-to-IPv6 transition) suppressing ICMP messages. The suppression of ICMPv6 messages (especially ICMPv6 "Packet too big") means that end systems are not automatically notified, when the PMTU of a flow changes[1] and cannot readjust the size of the packets they send out[2]. Hence, a Fault-Removal action for this problem would be to reconfigure the firewall on the router in question, as to allow ICMPv6 messages according to the recommendations in [DKS07].

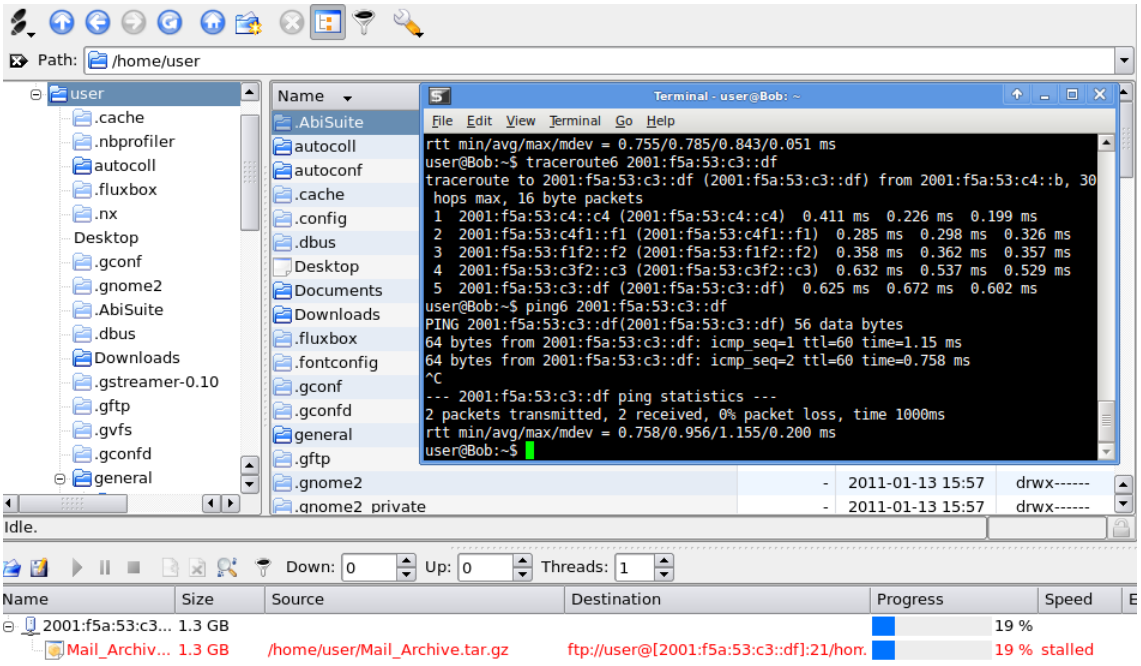

Figure 13.1: IPv6 Black Hole affecting an FTP Upload

The above described type of Black Holes may be extremely difficult to diagnose using traditional tools such as *ping* [PIN17] for reachability checks and *traceroute* [TRA17] for traffic path analysis. Figure 13.1 illustrates the transient nature of the Black Hole problem. On the one hand, the FTP upload is stalled and its packets are getting lost on a Black Hole router because of a sudden change of the PMTU and suppressed ICMPv6 "Packet too big" messages on the Black Hole router. On the other hand, a reachability test from the client system to the FTP server is successful (see the shell terminal on Figure 13.1), because of the small size of the used ICMPv6 packets to test reachability. However, if FDH is correctly introduced and configured within a network, it can

[1] A change of the PMTU is illustrated in the following

[2] Recall that packets do not get fragmented on intermediate routers in IPv6

remediate such issues and remove the fault in the Black Hole router. This is demonstrated within the next section.

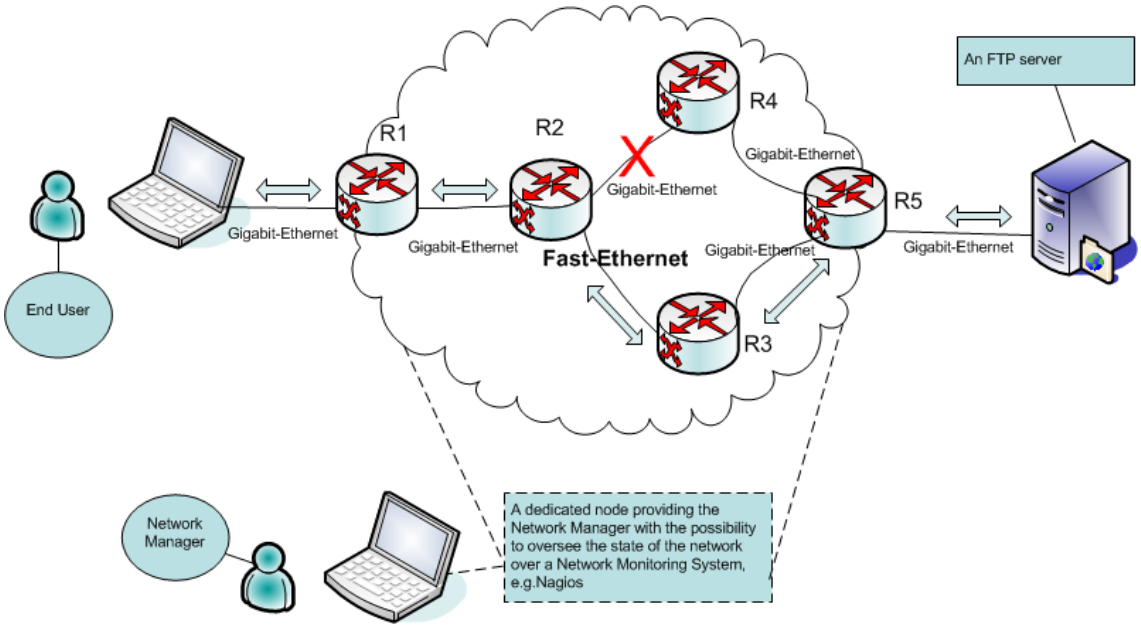

Figure 13.2: Reference Network for the IPv6 Black Hole Case Study

13.1.2 Resolving PMTU Black Holes in IPv6 Networks

The above discussed Black Hole erroneous state appears after a change in the PMTU towards a host during the lifetime of a flow. This PMTU change will mostly be caused by a link failure and automatic rerouting (as done in OSPF) over a link with a smaller MTU. Referring to the network in Figure 13.2, FTP upload type of traffic was pushed over the path *End User* → *R1* → *R2* → *R4* → *R5* → *FTP server 1*. After emulating a link failure between R2 and R4, the traffic was rerouted over the R2 → R3 which led to a decrease in the PMTU of the FTP upload. This decrease resulted from the fact that the path before the link failure was a pure Gigabit Ethernet path, i.e. with link MTUs greater than 1500 bytes, and the path after the link failure included the R2 → R3 Fast Ethernet link with a maximum MTU of 1500 bytes. Hence, for the case study experiments, R2 had a misconfigured firewall suppressing ICMPv6 "Packet too big" messages. Without the FDH mechanisms, this always resulted in a stalled FTP upload despite the fact that network connectivity between *Bob* and *FTP server 1* was present (see Figure 13.1). A possible solution for the above described issues could be the deployment of TCP PMTU discovery on the end system which is often not the case for standard Linux and Windows end systems. Nevertheless, FDH could help resolve this type of problems and prolong the lifetime of the connection in the above context. Within the conducted experiments, this was achieved by installing monitoring sensors (orchestrated by the MonAgent) on *R1*, which were monitoring for different traffic anomalies on the interface towards the end system and were reporting these anomalies to the FDH instances running across the network. More details on these aspects are given in the following paragraphs.

One of the key aspects that has to be handled at the start is given by the construction of the required Fault-Propagation Model facilitating the process of Fault-Isolation. [Lah00] provides packet flow descriptions that illustrate an IP Black Hole as observed on an intermediate router, i.e. a router be-

tween one of the end systems and the black hole router. Such a router would be *R1* in Figure 13.2, because R2 is considered as the black hole router. [Lah00] illustrates a case when large packets fail to traverse the network. However, since it is difficult to precisely define what a large packet is, the anomalies of *TCP duplicate acks* and *increased TCO retransmissions* as well as *sudden drops in non-complete (i.e. no FIN packet observed) TCP flows* were additionally considered as symptoms for an IP Black Hole[3]. Once these anomalies were detected, the corresponding incident descriptions were locally stored in the FDH incidents repositories on R1, and subsequently sent to R2, where they were processed according to the schemes on Figure 6.10 and Figure 6.13 in section 6.3. As previously mentioned, the key process of in the above referred dynamic aspects from section 6.3 is the process of Fault-Isolation that relies on a Fault-Propagation Model, which must capture the fault-propagation chain for a potential Black Hole within the network on Figure 13.2. In order to define a concrete fault-propagation chain, the following incident events were defined:

- **IPBH_Fault_1:** Misconfigured firewall suppressing ICMP packets on R2.
- **IPBH_Error_1:** R2 is unable to send ICMP messages.
- **IPBH_Error_2:** R2 fails to send ICMPv6 Packet Too Big notification to the corresponding end system.
- **IPBH_Error_3:** Incorrect PMTU assumed on the end system.
- **IPBH_Error_4:** Large TCP packets fail to pass R2.
- **IPBH_Error_5:** Duplicate ACKs in the direction[4] from R1 into the network.
- **IPBH_Error_6:** Duplicate ACKs in the direction[5] from the network to R1.
- **IPBH_Failure_1:** Duplicate ACKs observed on R1.
- **IPBH_Failure_2:** Increased number of TCP retransmissions observed on R1.
- **IPBH_Failure_3:** Sudden drops in the TCP connections observed on R1.

These are just short descriptions of the events. In the implementation of FDH, these incident descriptions are represented by alarm incident models as defined in [Tch09]. Figure 13.3 illustrates the corresponding fault- propagation chain, which is to consider when defining a Bayesian Network or Markov Chain based Fault-Propagation Model for resolving the IP Black Hole problem in the case study network (Figure 13.2). The belonging Fault-Propagation Model facilitated the successful FDH based diagnosis of the IPv6 Black Hole on R2.

Finally, the Fault-Removal action according to the scheme on Figure 6.13 consisted of removing "Packet too big" ICMPv6 suppression from the *R2* firewall in compliance with the guidelines given in [DKS07]. This was allowing the FTP upload to continue and influenced positively parallel UDP transmissions traversing the affected path. Furthermore, given that the end system has also deployed the FDH software, it was possible to trigger a Fault-Mitigation reaction to readjust the PMTU assumed on the end system (in line with the behavior specified in Figure 6.11), thereby

[3] In the next section, it will be seen that this can also be considered as a symptom for Ethernet Duplex Mismatch and that the result of the Fault-Isolation depends on the probabilities in the employed FPM

[4] This is the case of a TCP flow in which the end user system is sending packets to the FTP server, which is generating duplicate ACKs in order to indicate packet loss.

[5] The inverse case of the previous footnote, i.e. the FTP server is sending packets to the end system.

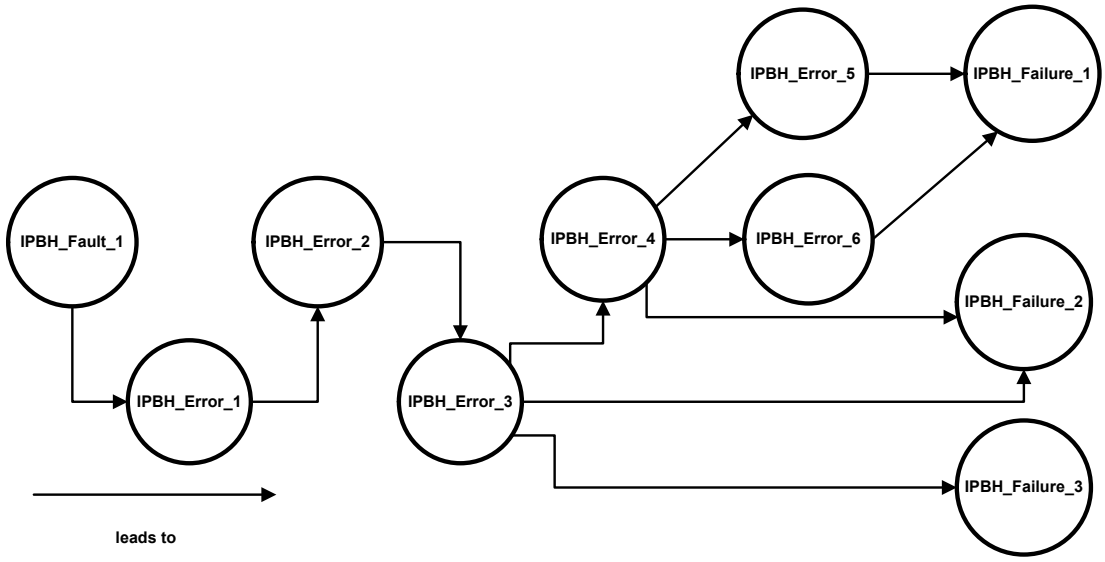

Figure 13.3: The Fault-Propagation Chain for the IPv6 Black Hole Case Study

successfully handling the present shortcoming due to the missing TCP PMTU discovery on the end system. This was especially enabled by the true end-to-end paradigm of IPv6 allowing transparent communication with the involved end systems and reaching them directly over an IPv6[6] address, i.e. without any intermediate NAT boxes.

The case study presented here shows that indeed the principles of FDH are implementable and once provisioned, can enable self-healing features within the network and the devices. In addition, it also demonstrates how FDH based self-healing mechanisms can extend the capabilities of IPv6 to overcome ICMPv6 suppressions in the ISP core network.

13.2 Failed Auto-configuration in IEEE 802.3

The second case study[7] is based on emergent faulty conditions in an IEEE 802.3 Ethernet environment. IEEE 802.3 Ethernet networks are an established communication environment for a variety of networks, e.g. LAN and Enterprise Networks. First, the problem under consideration is described, followed by a description of the tests which were conducted with automatically resolving the belonging fault by means of FDH. Some aspects of this case study were already used for illustrating the single FDH processes (e.g. Fault-Isolation) in the belonging sections. For that reason the following paragraphs describe the networking problem on the surface, whilst focusing on the overall process of self-healing with respect to a failed auto-configuration in IEEE 802.3.

The current case study fits to the type of faulty conditions presented in section 5.3.1 and section 5.3.6, i.e. *Faults/Errors/Failures detected at the Network Layer, Link and Physical Layers* and *Emergent Behaviors resulting from Software/Hardware Errors, Interoperability, Performance, Security related, and Automation Issues*. The classification results from the fact that the presented Auto-configuration problem in IEEE 802.3 results from design and quality assurance flows in the IEEE 802.3 auto-negotiation procedure, which impact the link and network layer of the network

[6] As previously mentioned, the testbeds used within this thesis are based on the IPv6 protocol.
[7] Parts of the current case study were initially published in [WTVL13].

architecture in question.

13.2.1 Auto-Configuration in IEEE 802.3

As discussed in the previous chapters, IEEE 802.3 Ethernet supports an auto-negotiation procedure that allows the Ethernet interfaces on the involved nodes to automatically obtain and establish suitable parameters for a link. The aforementioned parameters include the speed of the interface cards (e.g. 10 MBit/s, 100 MBit/s, 1000 MBit/s), the duplex mode, as well as flow control information. Whilst this procedure makes the setting up of a network easier, it can lead to a duplex mode mismatch in the settings assumed by both involved NICs (Network Interface Card). Such duplex mismatch can potentially slow down network paths and cripple the performance of the network. This anomaly has been extensively studied in [SC05] where also the symptoms and the chains of events leading to those symptoms are described in detail. [SC05] gives the following high level explanation of the conditions leading to a Duplex Mismatch situation on an Ethernet link:

"A duplex mismatch can happen in one of these ways (among others), but the symptoms seen by the hosts will be the same regardless of the cause:

1. *"One card is hard-coded to use full-duplex and the other is set to autonegotiate: the hard-coded side will not participate in negotiation and the auto-negotiating side will use its half-duplex default setting;*
2. *"The two cards are hard-coded to use different duplex modes;*
3. *"Both cards are set to auto-negotiate, but one or both of them handles autonegotiation poorly" ... "; note that, in this case, the problem can occur sporadically and rebooting or resetting the interface on either side could clear the condition."*

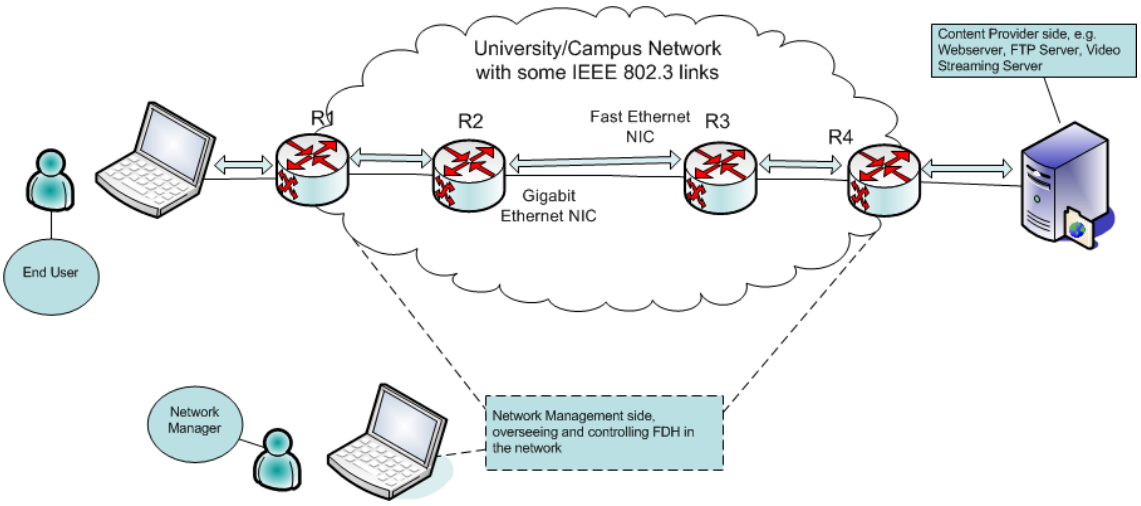

Figure 13.4: Network for the Ethernet Duplex Mismatch Case Study

The case that is interesting from FDH's perspective is the third point above[8] By monitoring for the events defined later on and correlating the observations, an FDH running node should be able

[8]Tools such as **mii** [MII00] can be easily employed for fault injection, in order to facilitate the testing of FDH for this case study.

to diagnose Ethernet Duplex Mismatch and to automatically reset/reconfigure the concerned interfaces.

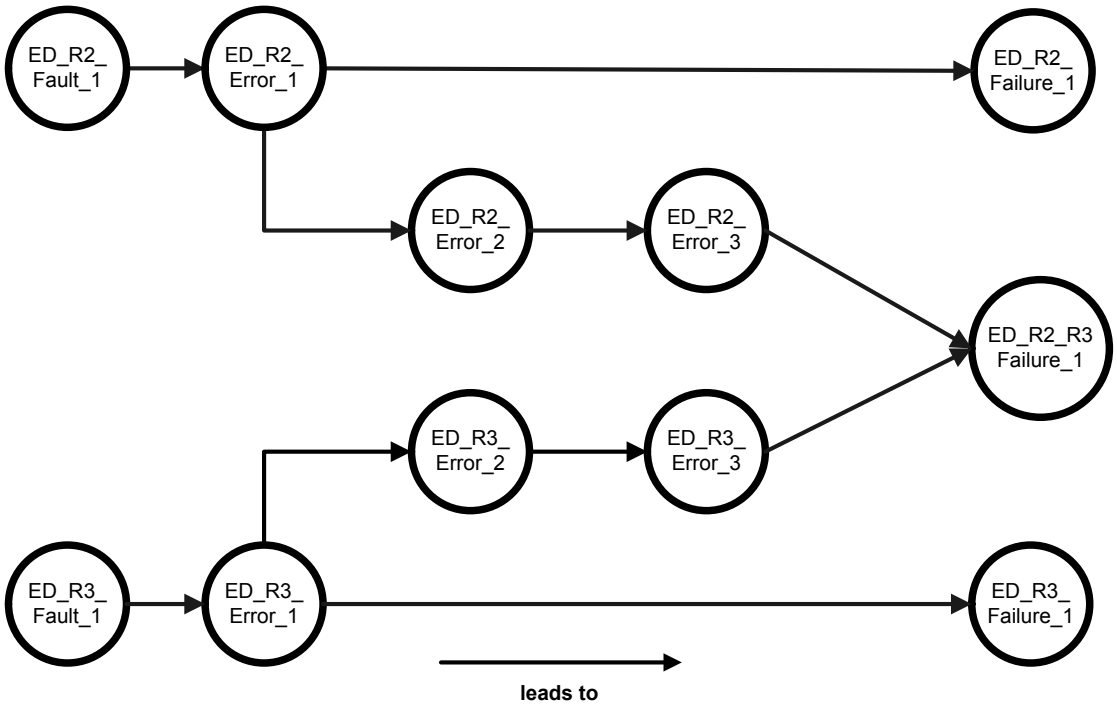

Figure 13.5: Fault-propagation Chain for Ethernet Duplex Mismatch

13.2.2 Resolving Duplex Mismatch in IEEE 802.3

Figure 13.4 presents the reference network for the current case study. This network is an example for a university/campus LAN and has one link (R2 ↔ R3) that has a Fast Ethernet and a Gigabit Ethernet card on its endpoints. Given a mismatch of the duplex mode, the instrumented FDH mechanisms detect the symptoms, isolate the fault, and set both cards in full duplex mode.

As in the previous case study, the Fault-Isolation process is the most challenging one requiring a Fault-Propagation Model that allows to infer the correct root cause based on the observed symptoms. In this line of thought, the network administrator would analyze existing publications, or would be just experienced enough to recognize the potential for an Ethernet Duplex Mismatch on the R2↔R3 link. Material for constructing the fault-propagation chain for Ethernet Duplex Mismatch in the Figure 13.4 network is provided in [SC05]. Based on this information, the following set of potential incident events were identified, which are used to compile the fault-propagation chain in Figure 13.5:

- **ED_R2_Fault_1:** The NIC on R2 operates mistakenly in half-dulpex mode.
- **ED_R2_Error_1:** R3 fails to send some frames to R2 because of interferences with the R2 NIC mistakenly operating in half-duplex mode (for more details see [SC05]).
- **ED_R2_Error_2:** TCP packets flowing from R3 to R2 get lost.

- **ED_R2_Error_3:** Duplicate ACKs in TCP flows streaming over R3 and R2 in a row.
- **ED_R2_Failure_1:** Increased number of dropped packets at the R3 NIC connecting R2 and R3
- **ED_R3_Fault_1:** The NIC on R3 operates mistakenly in half-dulpex mode.
- **ED_R3_Error_1:** R2 fails to send some frames to R3 because of interferences with the R3 NIC mistakenly operating in half-duplex mode (for more details see [SC05]).
- **ED_R3_Error_2:** TCP packets flowing from R2 to R3 get lost.
- **ED_R3_Error_3:** Duplicate ACKs in TCP flows streaming over R2 and R3 in a row.
- **ED_R3_Failure_1:** Increased number of dropped packets at the R2 NIC connecting R3 and R2.
- **ED_R2_R3_Failure_1:** Increased number of duplicate ACKs in TCP connections flowing over R2 and R3.

The executed experiments consisted of introducing series of duplex mismatches on the link R2↔R3 and monitoring the behavior of the FDH components in the routers towards a successful fault resolution. Figure 13.6 shows a snapshot of the Nagios plug-in output that allows the network administrators to observe the reaction of the Fault-Management control loop on different routers. The symptom of *duplicate acks in TCP flows* has been monitored on R1 (the edge of the network) and after correlating this symptom with the *increased number of dropped packets on the R3 NIC towards R2*, the deployed FPM - based on the fault- propagation chain on Figure 13.5 - facilitates the FaultManAgents to conclude that the R3 NIC connecting to R2 is the one that has mistakenly auto-configured to operate in half-duplex mode. Consequently, the FaultManAgents employed the ethtool [ETH05] command line interface to set the correct duplex mode. During the experiments it was observed that the FDH based Fault-Removal of duplex mismatch faults led to reduced number of retransmissions and duplicate acks (i.e. out of order packets) in TCP flows. Indeed, the distributed collaboration of the FaultManAgents in the nodes successfully improved the overall quality of service provided by the case study network.

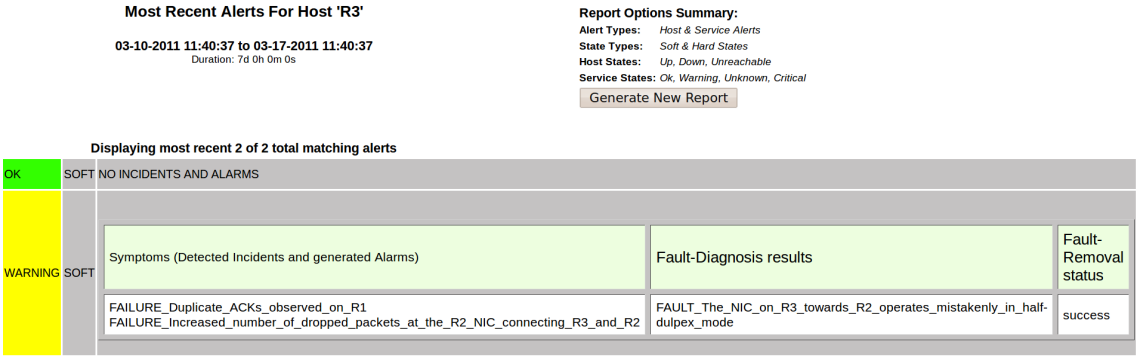

Figure 13.6: Ethernet Duplex Mismatch - Nagios

13.3 Architecture for Distributed Intelligence in an IP-based Surveillance System

The following case study has emerged within a scientific collaboration between Fraunhofer FOKUS and the VTT Technical Research Centre of Finland. Thereby, the predecessor of the self-healing architecture presented in this thesis was theoretically applied [TGV09] for increasing the resilience of the SPY architecture for mobile wide-area surveillance. The latter is one of the research topics [NRL09] at VTT. In the course of this collaboration, various potential problems[9] within the SPY architecture were identified and the way these would be handled by the FDH framework investigated. In the following, the SPY architecture is briefly described, which is the prerequisite for presenting the potential problems and determining the way FDH would increase the SPY resilience with respect to these potential issues.

13.3.1 The SPY Architecture for Mobile Intelligent Surveillance

SPY (Surveillance imProved sYstem) is an architecture for distributed, surveillance of a physical area, whereby the communication between the various components runs on top of an IP network technology. Thereby, SPY relies on the use of both distributed and centrally coordinated intelligence, as well as on the use of mobile units for exploring the area under surveillance. A simplified deployment of the SPY architecture is depicted in Figure 13.7. This deployment consists of an arbitrary number of mobile units equipped with intelligent sensors, as well as a centralized control room, which reacts to alerts and manages the mobile units.

The mobile units communicate with the control room side using wireless IP-based networks, e.g. UMTS, LTE or HSDPA. Control room components would be normally connected among each other using Ethernet with an IP stack on top of it. The core of a Mobile Sensor Unit (MSU) (on the left on Figure 13.7) is constituted by a piece of software on-board of a mobile unit deployed in the area under surveillance. This software processes data from one or more intelligent sensors (e.g. intelligent video cameras and audio sensors), which provide event data or metadata, such as "suspicious object detected in video", instead of, or in addition to raw sensor data. Furthermore, a GPS receiver is required on each MSU in order to periodically report the unit's position to the control room. The key functionality of the MSU software is given by its decision making capabilities for processing event input from sensors. In the scenarios presented here, the decision making of the MSU plays a role for two purposes:

- filtering out redundant or benign events reported by sensors, in order to save wireless network bandwidth and control room processing time, and
- combining together multiple sensor inputs and that way detecting events, which are to be reported to the control room.

Proceeding with the components on Figure 13.7, the *Gateway* is responsible for dispatching messages between the MSUs and the control room components. That way MSUs are required to know only the IP address of the Gateway and should not be aware of the complexity of the control room architecture when delivering their information. Furthermore, the control room components do not

[9] In that case, the FDH was also applied to handle problems within the mobile part of the SPY architecture. Hence, showing that despite being focused on fixed network structures, the FDH can also be utilized for improving the resilience of mobile devices/nodes in some special cases.

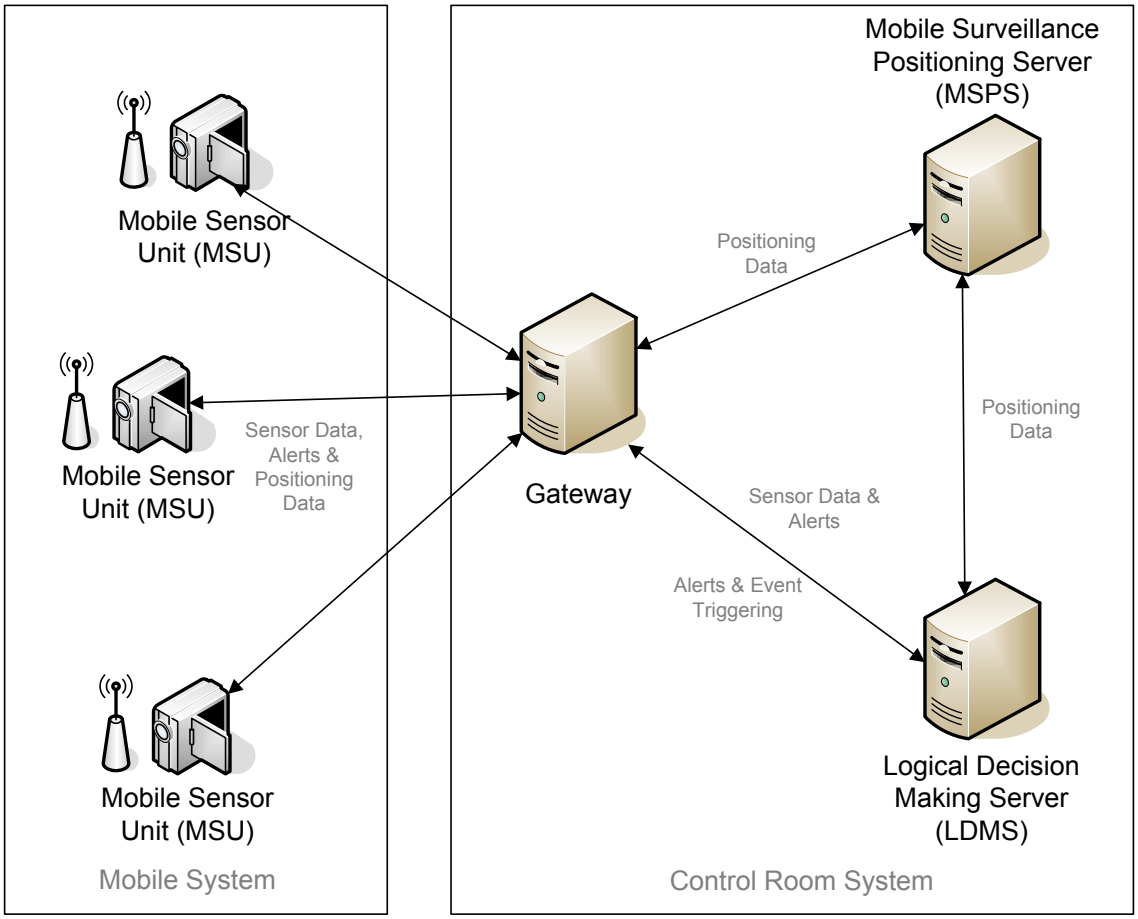

Figure 13.7: SPY Architecture for Mobile Intelligent Surveillance as described in [NTSR14]

have to maintain permanent connections to each of the MSUs currently online in the system on a single basis (i.e. for each component). The Mobile Surveillance Positioning Server (MSPS) (on the top right in Figure 13.7) handles the positions of the MSUs, which are periodically reported by the MSUs over the Gateway. Thereby, the MSPS offers an interface for the other control loop components to query the positions of the involved MSUs. Finally, the Logical Decision Making Server (LDMS) provides functions for automatically utilizing and analyzing data from the MSUs. It is based on previous VTT work related to a Single Location Surveillance Point System [NRL09]. The MSU data analysis involves aspects such as correlating reports from multiple MSUs into a single event, deciding on the significance/seriousness of the event and the need for an action based on the correlation results. The actions that could be issued by the LDMS include sending commands to the MSUs - e.g. directing a unit to relocate to the area of the perceived threat, or generating alarms for the surveillance personnel. Having briefly presented the SPY architecture, the following sections identify potential network/infrastructure challenges that may occur during the SPY operational phase, and describe the way FDH would handle these resilience challenges.

13.3.2 Potential Problems resolvable by FDH Means

This section presents two potential problems with the goal to illustrate the interplay between the SPY mobile surveillance architecture and the FDH framework for distributed self-healing. Thereby, the following two scenarios show how the robustness of the surveillance system is increased, which in turn allows for better managing complex emergency situations and can support lifesaving operations. Hence, it is presumed that all nodes (MSUs and control room components) are equipped with an FDH instance.

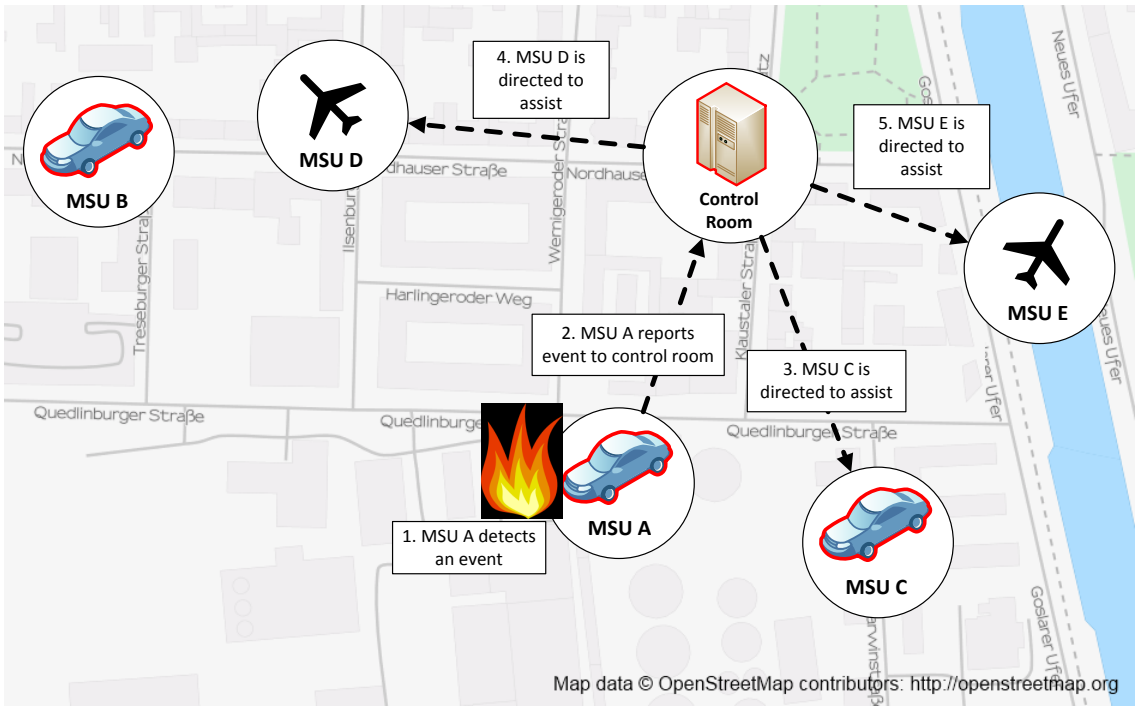

Figure 13.8: First SPY-FDH Case Study Illustration

13.3.2.1 Overcoming Gateway Unavailability through FDH based Self-Healing

The surveillance use case for the first scenario is depicted in Figure 13.8. The use case[10] includes five patrolling MSUs, which are labeled as MSU A through to MSU E. MSUs A to C represent road vehicles, while MSUs D and E are UAVs (Unmanned Aerial Vehicles) equipped with cameras for overseeing the situation. Based on these prerequisites, the scenario consists of MSU A detecting an alarming event and reporting it to the control room that in turn concludes that there is a need for additional MSU support. Based on the MSU's positioning data, nearby patrolling units are instructed to assist. In the situation depicted on Figure 13.8, this would mean that road vehicle MSU C in addition to UAVs D and E is instructed to approach the coordinates of the alarming event.

Assuming that the initialization process of the SPY components has been successfully accomplished - i.e. all MSUs have successfully registered at the LDMS in the control room and are periodically reporting their position to the MSPS, the specific steps for the current case study are depicted by the sequence diagram on Figure 13.9:

1. MSU A sends a message to the LDMS, reporting a sensor event determined to be a potential fire incident (1).
2. The LDMS decides to instruct other MSUs to provide assistance based on the coordinates and geographical proximity of these MSUs.
3. The LDMS redirects MSU C to move to the coordinates of the potential fire incident as being one of the MSUs which are close to the reported event, and finally
4. MSU A is notified of the additional MSU C, which has been directed to the area of the reported alarm (4).

Given a situation in which the burning building is also the one hosting a base station, the MSUs will experience a loss of network connectivity to the Gateway and correspondingly to the LDMS and MSPS components. This would severely hamper the functioning of the SPY framework and might even cause the loss of human life, since the rescue force would miss valuable information. Therefore, the FDH components in the SPY nodes must react in an attempt to restore the flow of messages and enable the transmission of events to relevant decision makers.

As mentioned before, this case study and belonging setup exemplify on how the FDH can be utilized for improving the resilience of mobile devices/nodes in a special case. This shows that the FDH can be also applied beyond the scope of fixed network structures, which are the main focus of the current thesis.

The reaction of the FDH components embedded in the SPY nodes consists of the following steps:

1. Once the SPY software on the MSUs detects the lack of network connectivity to the Gateway, it reports this incident to the local FDH repositories for faults/errors/failures/alarms.

[10] The presented scenario maps to section 5.3.1 and section 5.3.3 from the classification of faulty conditions, i.e. *Faults/Errors/ Failures detected at the Network Layer, Link and Physical Layers* and *Traditional Network and Systems Management Alarms*. This comes from the fact that the described scenario deals with an infrastructural problem - outage of a communication component - and would normally be handled by the network operations personnel in the long term. In this context, FDH provides the means to overcome the network component outage and facilitates handling the emergency situation in question.

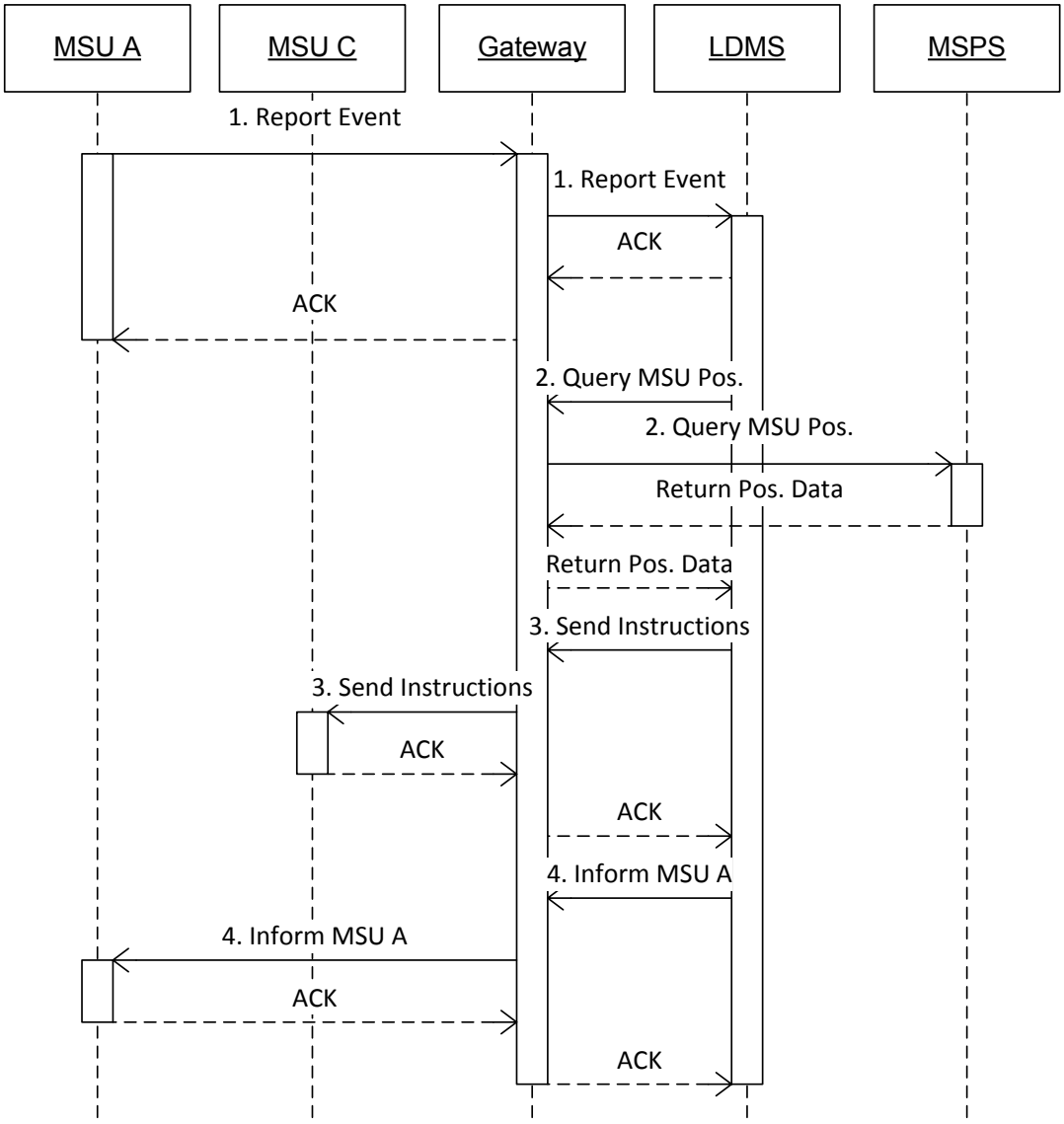

Figure 13.9: The Interaction Flow between the SPY Components after an Alarm Report within the Case Study

2. These incident descriptions (in each MSU) are then forwarded to the belonging local FaultManAgents and to the local SurvAgents within the SPY nodes.

3. In addition, each incident description is submitted to the local Event Dissemination Agent of the FDH instances in the SPY nodes, with the goal to convey it to the largest possible number of MSU nodes.

4. Subsequently, each EvDissAgent broadcasts the incident description over the radio link and disseminates it that way to the other involved MSUs.

5. Hence, after some time all the involved MSUs (A, C, D, and E) will have informed each other about their lack of connectivity to the Gateway.

6. Based on this set of incident descriptions, the Fault-Isolation Functions (FIF) inside the FaultManAgent of each MSU conclude that there is a severe failure of the corresponding base station.

7. In order to overcome this challenge, the FaultManAgents of each MSU need to react as to ensure the delivering of messages to the Gateway. In that context, the FRF within the FaultManAgents of all road vehicles are pre-configured as to influence the network stack on the belonging MSU to stop sending information to the mobile network (LTE or UMTS), but instead to just broadcast it to other MSUs over the radio link.

8. In addition, the FRFs on all UAVs are pre-configured as to switch to a store-carry-forward (SCF) mode as discussed in [Fal03] [SHÇ+10]. In such a mode the UAVs would receive and store a number of packets sent out by the road vehicles, carry those packets to the next functioning base station, and off-load them to the network towards the Gateway.

Steered by the FDH components in each MSU, data flow is maintained from the MSUs to the Gateway and correspondingly to the LDMS and MSPS. Certainly, it is impossible to transmit continuous data in such a situation. However, in that context, the SCF method would facilitate the reporting of non-continuous data, such as new events and images, to the Control Room, thereby allowing relevant decision makers to better take action.

13.3.2.2 Self-Healing within the IT Infrastructure of the Control Room System

The second case study is related to the IT infrastructure running in the Control Room System and the use of database technology in that context, as illustrated in Figure 13.10. Hence, it fits to section 5.3.2 (i.e. *Faults/Errors/Failures detected at the Services and Application Layers*) from the classification of faulty conditions in chapter 5. Furthermore, given the nature of the presented problem, it can be also classified and mapped to section 5.3.5, which is described as *Unknown Faults resulting from insufficient Testing of Software*.

During the operation of the SPY software, regular interactions and queries are required between the MSUs and the MSPS, in order, in order to handle the positioning data for the involved MSUs. All these communication to and from the MSPS implies regular updates and queries issued on the database running in the background. A possible implementation is given by a setting in which the MSPS uses a Hibernate [Hib17] persistence layer to communicate with a PostgreSQL [Pos17] database (see Figure 13.10). Practical experiences with such a setting have shown that a version

mismatchbetween the Hibernate database driver and the PostgreSQL database can lead to a problem in the closing of database connections[11]. This problem might in turn sporadically lead to failures while opening new database connections, and correspondingly to problems querying and updating MSU positions within the MSPS. Assuming that the MSPS developers and operators are not aware of the version mismatch, they would only observe that database connections are sometimes not released resulting in transient problems while handling GPS data. In order to remediate this problem, the MSPS operators would use the embedded FDH self-healing capabilities and features. That is, a lightweight monitoring daemon/sensor would be orchestrated by the MonAgent on the MSPS node, in order to observe the number of open database connections. This daemon would interplay with the other FDH agents - e.g. with FaultMangAgent and the SurvAgent - on the MSPS node in the following way:

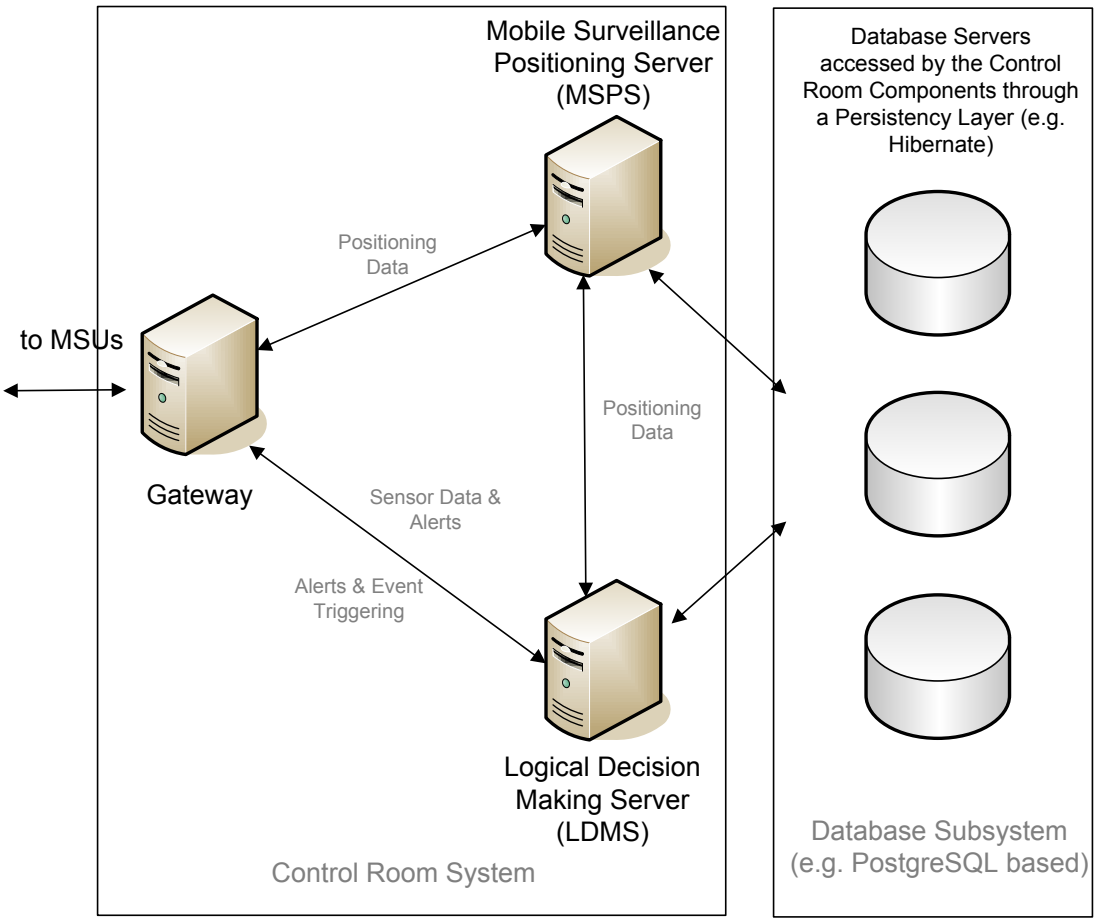

Figure 13.10: VTT Control Room Architecture with Database Subsystem

1. The MonAgent database connections monitoring plug-in would periodically check the number of open connections.

2. In case a pre-defined number of open connections is being approached, the monitoring plug-in would create an alarm description, which would be pushed to the FDH repositories on

[11] It might be a difficult task to identify the version mismatch as responsible for the problem of database connection closing, since in general the entire set of database features function, and the failure to close database connections appears non-deterministically. In the scope of different projects, a couple of months and a large number of tests were required to diagnose such a problem in a (pre-)product.

the MSPS node.

3. The MSPS-node-EvDissAgent would in turn disseminate the alarm description to the involved machines and would correspondingly notify the FDH components on the machine hosting the database.

4. Subsequently, the alarm description would be forwarded to the FMF functions of the SurvAgent on the machine hosting the database. These FMF functions are pre-configured as to react by automatically restarting the PostgreSQL database and that way releasing the large number of open connections.

5. A restart is also required for the database communicating module of the MSPS, in order to fully release the connections which were opened on the MSPS node. Therefore, the FMF functions of the belonging SurvAgent would automatically restart the MSPS server and trigger the release of all database connections.

By executing these steps, the MSPS services would be only shortly unavailable and would then continue handling GPS data from the MSUs. Hence, by introducing FDH based self-healing features into the SPY architecture, it is possible to remediate emergent erroneous situations in the IT infrastructure of the Control Room system, e.g. due to the unawareness of the MSPS operator for a version mismatch between a Hibernate driver and a PostgreSQL database. Such an FDH self-healing "patch" would help to sustain operations whilst the experts analyze the problem and find a suitable solution in the long term.

13.4 Combined Case Study

The final case study demonstrates the capability of FDH to remediate multiple parallel challenges in an ISP network. Thereby, this combined case study builds on some of the case studies presented in the previous subsections and adds some additional potential problems/faults within an ISP network. Indeed, the key potential problems were selected based on a survey that was conducted in the course of a large scale European project, related to the topic of combining IPv6 and Autonomic Computing towards improving various network management aspects.

13.4.1 Case Study Description

The FDH framework was applied to a combined case study, involving a combination of different networking problems being introduced to the experimental IPv6 network in Figure 11.1. As previously mentioned, this network was designed in the course of the EU EFIPSANS integrated project [efi16a] based on the requirements which were consolidated from the participating industrial and academic partners.

13.4.1.1 Networking Problems Survey

In addition, the partners within EFIPSANS have conducted a survey on potential networking problems, which provided the base for the combined case study presented here. Thereby, the focus was set on problems that can occur in IPv6 networks, as the testbed in Figure 11.1.

The survey involved a two-stage process, the first part being an analysis by the consortium of problems encountered in the course of installing and operating IPv6 - both from their own experience as part of the project, and their knowledge of the industry at large. The second stage involved an analysis of the IPv6 protocols involved in these issues by understanding the IETF RFCs related to each protocol and an analysis of how each fault occurs and their likelihood. Additionally, a number of papers, books, and (also non-IPv6) RFCs were examined in order to identify problems suitable for demonstrating the concepts proposed by EFIPSANS. The types of problems the EFIPSANS consortium was looking for include:

1. Misconfigurations due to hacker attacks or due to the adoption of configuration profiles from another network - e.g. parts of an IPv4 configuration inherited after the network has switched to IPv6
2. Maintenance activities - according to [Tch10] and EFIPSANS industrial partners such interventions account for 20% of all failures
3. Overload of particular nodes or paths
4. Emergent behaviors - i.e. situations in which each functional entity works correctly according to its specification, but the overall set of behaviors leads to performance degradation of the overall system
5. Well studied anomalies, which can be potentially remediated by networking management automations as investigated within the EFIPSANS project, and
6. Incorrect behaviors of functional entities due to *software bugs* or *hardware defects*

These criteria/constraints, according to which the EFIPSANS consortium conducted the survey, can be directly mapped to some parts of the broader classification of faulty conditions, which was presented in section 5.3.

As a result of this survey, some problems (faults) were selected according to:

1. The impact of the faults on the IPv6 stack - given the IPv6 based testbed in chapter 11 and Figure 11.1
2. The likelihood of these faults occurring in industry networks
3. The relevance of these faults to IPv6 and the EFIPSANS project - given that the testbed was of relevance for this EU-project

The study provided insights and inspirations for the FDH combined case study that was worked out in the course of this thesis.

13.4.1.2 Setup Description

The following paragraphs elaborate the key aspects of the experimentation setup starting from the network setup and the concrete problems, which were (periodically) introduced to the testbed in the course of the combined case study, and proceeding with key configurations for the FDH components deployed across the testbed nodes, such as Fault-Propagation Model, monitoring sensors, and various required reaction policies.

Infrastructural and Traffic Considerations: Before diving into the details of the FDH configurations and the emulated network problems, a general remark on the case study setup is needed: The presented case study emulates problems within an ISP network. Thereby, the routers *R1-R11* play the role of the nodes within this ISP network. Furthermore, the client end systems - *Alice*, *Bob*, *Charlie*, *Dave*, and *Paul* - play the role of subscribers to the ISP's network. Thereby, in order to focus on the IP layer aspects, the various types of access network components, such as DSLAM, Diameter/RADIUS server, etc, are omitted. Moreover, the machines on the right side of Figure 11.1 are considered as OTT (Over-the-Top) services[12], such as video streaming platforms, FTP or Webservers. Furthermore, the ISP's network provides basic networking services like DNS, which is available in the *S3* network in the middle. In addition, it is presumed that the ISP offers a storage service for its subscribers, which is realized by an additional *FTP* server in the *S3* network in the ISP's domain.

In order, to create an environment for testing the FDH behavior, *iperf* [IPE17] and *ping* [PIN17] traffic generators were started on the client end systems (Alice, Bob, Charlie, Dave, and Paul) on the left side. These generators were pushing traffic towards the OTT servers on the right side as well as towards the FTP server within the ISP's network. Thereby, TCP and UDP traffic (based on *iperf*) and ICMPv6 traffic (based on *ping*) was pushed between the client end systems and the servers.

Concrete Network Problems: For the sake of the combined case study, the various networking problems (inspired by the above survey) and corresponding faults were injected by an automated script that was issuing commands on the network nodes on Figure 11.1 over the management interfaces. In particular, four different types of problems were introduced at multiple points in the network including:

1. Black Holes ([Lah00] and section 13.1) in IPv6 on R9 and R1
2. Ethernet Duplex Mismatches ([SC05] and section 13.2) on links R9↔R10 and R10↔R11
3. large packet delays resulting in low quality of the link R1↔R2[13]
4. crash of the DNS (Domain Name System) server running in sub network S3 - thereby emulating the problem of a server crash with a serious impact on the ISP infrastructure[14].

In addition to the problems from the previous case studies (section 13.1 and section 13.2), the issues of an increased delay on a link and the crash of a DNS server are used in the case study. These circumstances might occur due to issues in the design and implementation of involved modules, e.g. memory leaks and bugs in the realization of link layer switches, drivers, or (DNS) server components. The DNS availability and the delay on a link are monitored by watchdogs that report incidents to the FDH machinery resulting in Fault-Isolation and eventually the restart of the server or the switching off of the NIC in question, such that the traffic avoids the link with the large delays. Within the experimentation flow, the delay on the R1↔R2 link is incrementally increased

[12]The term Over-the-Top denotes in general services which are not provided by an ISP (or a network provider in general) but rather by a content provider who maintains a set of servers and ideally a belonging CDN (Content Distribution Network).

[13]The problem can be seen as belonging to the class *Faults/Errors/ Failures detected at the Network layer, Link and Physical Layers* , i.e. section 5.3.1 from the classification of faulty conditions.

[14]This problem maps to section 5.3.2 from chapter 5 (i.e. from the classification of faulty conditions) and is described as *Faults/Errors/Failures detected at the Services and Application Layers* thereby understanding DNS as a network service.

which is detected by the monitoring watchdog orchestrated by the MonAgent in the context of the FPF module of the SurvAgent, thereby shaping a scenario for the Failure-Prediction aspects of FDH.

Fault-Propagation Model: The above listed potential issues result in an overall number of five network faults being injected by an automated script at different points (links, routers and servers) in the testbed network on Figure 11.1. Correspondingly, a Fault-Propagation Model comprising around 40 events was constructed as to facilitate the Fault-Isolation process in the network nodes. Thereby, the Fault-Propagation Model includes sub-models for Ethernet Duplex Mismatch and IPv6 Black Holes which are similarly constructed to the models presented in the previous sections 13.1 and 13.2. These sub-models reflect the particular fault-propagation chains for the specific problem, either an IPv6 Black Hole on *R9* or *R1* or a Duplex Mismatch on the link R9↔R10 or the link R10↔R11. Thereby, the above IPv6 Black Hole and Duplex Mismatch problems can manifest by some common observable errors/failures, according to the deployed Fault-Propagation Model and monitoring components, e.g. the failure of "Duplicate acks in TCP connections passing through R4" (a similar event would be also observed on R8) can be a consequence either from an IPv6 Black Hole on R9, or from an Ethernet Duplex Mismatch. In the course of that, the Fault-Isolation result is determined by additional events - e.g. "Sudden drops in TCP connections" (on a router traversed by affected traffic) for IPv6 Black Holes, or the "Increased number of dropped packets" at the corresponding machine in the case of any Ethernet Duplex Mismatch. These events influence the event correlation procedure as to narrow down to the most likely root cause. In addition, a fault- propagation chain that reflects the DNS server crash is embedded in the Fault-Propagation Model for the combined case study. Thereby, the domain name resolution of the service provider network is periodically checked on the edge router(s) close to the client end systems. In case of unsuccessful domain name resolution, several possible fault-propagations might have caused it, including Black Holes, Duplex Mismatches and a crash of the DNS server in the S3 network. The most likely root cause depends again on the correlation of the overall set of observed incidents. In case only the domain name resolution failure was observed as a single event, the Fault-Propagation Model was designed to lead to the direct conclusion that the DNS server must have crashed.

Fault-Detection and Monitoring: In order to detect different symptoms related to the above networking problems, a number of monitoring plug-ins where instrumented within the MonAgents across the network. In particular, monitoring plug-ins which were sampling traffic passing through the respective edge router - R1, R4, and R8 - were orchestrated by the belonging FDH MonAgents. These plug-ins were picking traffic from the passing TCP connections and analyzing it, in order to detect various anomalies which were indicating a problem in the ISP's network behind. That is the monitoring sensors were looking for "sudden drops (without an accomplishing FIN packet) in TCP throughput on an interface", "increased number of retransmissions in the TCP flows", as well as "increased number of duplicate acks in the TCP flows". Furthermore, monitoring "sensors" were installed on relevant edge routerrs as to control the DNS resolution service provided by the ISP network, as well as to detect the increase in the delay on the R1↔R2 link.

Failure-Prediction Functions: The Failure-Prediction Functions were instrumented as to anticipate problems on the link R1↔R2 based on the RTT delay measured between the two routers. An increase in such a delay can be the result of various root causes, such as transient errors due to bugs in the realization of the link layer switches or the NIC drivers on the involved routers - for instance memory issues leading to fatigue in the operation of the involved modules. As mentioned above, such increases in the RTT delay on the R1↔R2 link were tracked by a monitoring watchdog on R1 that was reporting to the FPF module of the SurvAgent on R1. Thereby, the FPF module was

using the exponential smoothing method that was described as a possible algorithm for Failure-Prediction in section 7.4.3. The exponential smoothing was giving an indication on whether there is a stable trend for an increasing low quality of the R1↔R2 link. Given that a particular threshold - resulting from the exponential smoothing average - was crossed, an alarm was generated by the FPF (see the sequence diagram in Figure 6.12) that was describing the low quality trend on the R1↔R2 link. Based on this alarm the Fault-Mitigation Functions were instrumented as to switch off the involved Network Interface Cards on R1 and R2. As a result the traffic was re-routed over the R1↔R4 link which is considered as the better option given the low quality of the R1↔R2 link.

Fault-Mitigation Functions: The Fault-Mitigation Functions were instrumented on various nodes across the testbed on Figure 11.1 whilst experimenting with the combined case study. Firstly, the R1 router had a Fault-Mitigation policy to switch off the link towards R2 in case of an alarm for a stable and continuous trend for low quality of the R1↔R2 link. This alarm is generated by the Failure-Prediction Functions as described above. Furthermore, the client end systems were equipped with a Fault-Mitigation policy that was allowing them to undertake an action upon getting notified of symptoms indicating IPv6 Black Holes in the network. These symptoms were given by "sudden drops (without an accomplishing FIN packet) in TCP throughput", "increased number of retransmissions in the TCP flows" and "increased number of duplicate acks in TCP flows". The action was constituted by setting the PMTU towards the corresponding servers to 1280 Bytes, which is the minimal MTU for an IPv6 network. That way, it was assured that the existing connections will proceed to serve the clients as long as the Black Hole problem is being removed in the network.

Fault-Removal Functions: With respect to the Fault-Removal Functions, various FRF policies were configured for the purposes of the combined case study. Starting with the IPv6 Black Holes within the case study, the reconfiguration of the firewalls on R1 and R9 (refer to the testbed network on Figure 11.1) was instrumented as a Fault-Removal behavior. Thereby, the firewalls on R1 and R9 were reconfigured as to stop blocking ICMPv6 "Packet too big messages" according to the recommendations in RFC4942 [DKS07]. This was implemented by the use of the *iptables6* [IP617] tool which is provided in Linux based operating systems. Furthermore, the fault of Ethernet Duplex Mismatch on the links R9↔R10 und R10↔R11 was removed based on preconfigured Fault-Removal Functions on the involved routers (R9, R19, and R11). Thereby, the *ethtool* [ETH05] was used in order to reinitiate the duplex auto-negotiation between the Network Interface Cards on the links in question, i.e. R9↔R10 and R10↔R11. Finally, a Fault-Removal action was configured on the DNS server machine in the S3 network - in the center of Figure 11.1. This action consisted of restarting the DNS server on the machine in order to restore its operation. Thereby, in case the DNS server has crashed, its shutting down was not needed, otherwise the restart was considered as a mean to remediate the fatigue in its operation leading to failed DNS query resolutions for the modeled ISP network.

In addition[15], different types of Fault-Removal assessment actions were configured for the FRAF modules of the FaultManAgents on the corresponding network nodes. These were checking on whether the firewalls were indeed successfully reconfigured, the domain name resolution was working correctly after a DNS server restart, or if the Duplex Mismatch on one of the IEEE 802.3 links (R9↔R10 and R10↔R11) was successfully resolved. In case these FRAF actions have identified a Fault-Removal action as unsuccessful, they would notify the network administrator over

[15] In the scope of the combined case study, the focus is set on demonstrating the ability of FDH to cope with multiple faults in a broader network environment. Therefore, the Self-Organization aspects were omitted during the case study design, given the need to go for an environment and set of problems which convincingly show the self-healing benefits. The self-optimization aspects were thoroughly analyzed and their benefits demonstrated in section 8.2.7.

the Operator Console on one of the servers in the S3 network in the center of Figure 11.1.

13.4.2 Qualitative Analysis

The qualitative analysis in this chapter builds on the setup and flows established in the combined case study from section 13.4.1. In that context, a number of trials with different series of the above[16] described networking problems (i.e. series of fault injections) were executed. Thereby, the various series were realized by automated scripts that were accessing the network nodes over the IPv4 management interfaces of the Figure 11.1 testbed. These scripts were automatically logging into the network nodes and reconfiguring them as to introduce the next networking problem. Subsequently, it was observed how the FDH mechanisms across the network nodes were implementing the belonging behaviors as to mitigate and/or resolve the introduced problems. In the cases where FDH failed to remove the introduced fault, the automated script for the trial was taking care of setting things back after some time, such that any after effects are removed before the next fault of the experimentation is introduced .

As mentioned above, a number of trials with different series of fault injections was executed. The ratio of successfully resolved faults is depicted in Figure 13.11. Thereby, the horizontal axis marks the different trials that were executed in turn, whilst the other axis shows the absolute number of successful fault-resolutions compared to the number of failed ones. In general the FDH managed to improve the resilience of the IPv6 network on Figure 11.1. There was one main reason why FDH could not remediate all introduced problems, namely the fact that the monitoring sensors were periodically sampling traffic, and thus sometimes missed to catch the incidents manifesting a Black Hole or Duplex Mismatch. However, the percentage of successfully detected, isolated and removed faults ranges from 50% to 94%, and on average 70% of the introduced faults were successfully resolved, such that the network could continue its normal operation.

To summarize: The quality experiments conducted within the combined case study setup (section 13.4) show that indeed the principles of FDH are implementable and once provisioned, can enable self-healing features within the network and the devices, thereby significantly improving the quality of a network[17].

[16] see section 13.4 for setup an networking problems of the combined case study

[17] In addition, the comibned case study and belonging qualitative experiments also demonstrates how the self-healing mechanisms of FDH can extend the capabilities of IPv6, e.g. to overcome ICMPv6 suppression based Back Holes in the core network

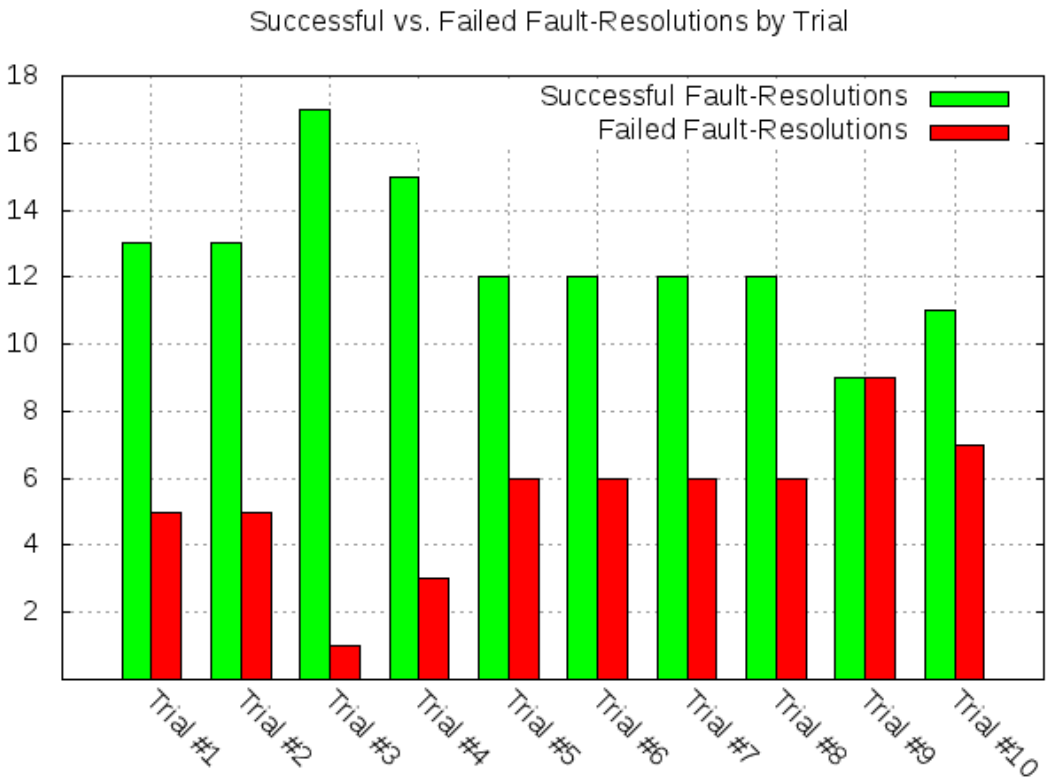

Figure 13.11: Number of successful Fault-Resolutions during the conducted Experiments

Chapter 14

Conclusions and Outlook

The following paragraphs provide the summary, conclusions and outlook of this thesis. First, a comprehensive overview of the presented research is given thereby outlining the key contributions of the single chapters and correspondingly the technical and scientific achievements. Subsequently, a discussion relating to the presented scientific output is provided. This includes an elucidation regarding how the problem statement of this thesis - formulated in the introductionary part - was addressed, followed by mapping the presented scientific work to the targeted research hypothesis, as well as showing how the pre-defined target objectives (as described in the current manuscript) are met by the presented research. Furthermore, a comparison to similar approaches and a general discussion regarding the role of the self-healing approach, which is at the heart of this thesis, is provided. Finally, future research directions are drafted and an outlook on possible further developments is given.

14.1 Summary

The current thesis deals with the topic of self-healing based on agents, which are distributed across the network devices and collaboratively take decisions regarding actions towards remediating erroneous states in a network/system. Thereby, the proposed agents are meant to act in a way as to optimize their overall reaction - i.e. resolve and synchronize (conflicting) actions of multiple involved agents towards increasing the overall performance and operations of the self-healing framework (consisting of the proposed agents), as well as of the network/system in question.

The proposed framework is meant to fill the gap between the network management layer and the network protocols itself with respect to handling faults/errors/failures/alarms in a networked environment. Traditionally, the network management substrate encompasses the processes of Fault-Management (according to the FCAPS TMN standard) that deal with aspects such as Fault-Detection, Fault-Isolation and Fault-Removal being executed in a standardized manner by the network operations personnel. Different tools, protocols, standards and methods are in place for facilitating the above listed tasks. These include monitoring protocols and techniques such as SNMP, NETCONF and RMON, as well as techniques and tools for Fault-Isolation - e.g. Bayesian Networks, Neural Networks, model based techniques and further. In addition, different protocols and methods (like CLI, TR-69, COPS, XACML ...) are available for reconfiguring network devices such that the processes of Fault-Management and correspondingly troubleshooting can be accomplished. All these techniques and protocols are implemented in various Operations Support-

ing Systems (OSS) across the network management layers - e.g. Element Management Systems (EMS) and/or Network Management Systems (NMS).

Looking from a different perspective; a number of fault-tolerant mechanisms are implemented in today's network protocols and services thereby realizing various vital aspects for increasing the resilience and survivability of networked infrastructures. Typical examples are given by the ICMPv4/v6 messages and mechanisms for network diagnosis and adaptation (e.g. PMTU discovery) as well as by the TCP mechanisms for congestion control and packet retransmission.

Considering the above perspectives, there is an obvious gap for establishing a framework that deals automatically with faults/errors/failures/alarms without every time involving the network operations personnel. Thereby, this framework is meant to take into account the intrinsic mechanisms within the protocols and services, catch the incidents which are not handled on that level, implement an immediate reaction in terms of Fault-Masking/Mitigation, analyze in parallel multiple incidents and come up with a strategy for resolving a malicious network state, which strategy is in turn to be executed in order to remove faults in the long term operation of the network. Furthermore, the framework should be intelligent and context-aware as to understand when it is not coping with the current network challenges, and should correspondingly escalate the situation to the network operations personnel thereby providing all required information for efficient Fault-Isolation and Fault-Removal according to traditional troubleshooting and Fault-Management principles.

Having identified the need for an agent based self-healing framework (incorporating self-optimization features), the thesis proceeds with an extensive analysis of scientific requirements towards the design and specification of the envisioned Framework for Distributed Self-Healing (FDH). This includes the identification of both function and non-functional requirements. Thereby, the functional requirements pertain to the functionalities required in order to fulfill the envisioned role of FDH in a distributed networked environment. These requirements help identify necessary components (e.g. agents, modules, libraries ..), processes/tasks, types of algorithms, and interactions between FDH components for the realization of scalable and efficient self-healing with self-optimization features. This includes requirements which lead to the key agents of FDH as well as to the main processes to be realized by the overall set of FDH agents across the network. Furthermore, the non- functional requirements mainly relate to the question of how the required functionalities are realized, thereby addressing aspects such as scalability, resource consumption and induced overhead (e.g. memory consumption, CPU utilization ...), dependability and stability of the self-healing control loops as well as security, to give some examples. Finally, different classes of problems from the area of networking and distributed systems are presented, which are defined as the target scope of FDH when it comes to automatically resolving erroneous states within modern and future networks/ systems.

Based on the above elaborated scientific requirements, the thesis proceeds with devising the key components of the FDH framework. Thereby, the focus is set on the components which operate within a single node with the presumption that components within single network devices will collaborate, in order to achieve self-healing for the network as a whole. Hence, the required processes inside a network node are described along the definition of the belonging FDH modules. Additionally, the self- healing and self-optimization tasks (comprising different required control loops) are specified on network level, whereby the direct and indirect collaboration between the network nodes is discussed on. This means that various required components are captured in the course of specifying the architecture of the FDH itself. Furthermore, the dynamic aspects of the FDH architecture are described, i.e. the interaction flows among the specified components on node and network level towards the realization of distributed agent-based self-healing with self-optimization

features.

The designed and specified components include agents such as **1)** the node level Fault-Management Agent - responsible for the automation of Fault-Detection, Fault-Isolation and Fault-Removal thereby realizing an automated Fault-Management control loop for achieving self-healing and fault-resolution in the long term operation of the network, **2)** the node level Survivability Agent - responsible for immediate response to a detected fault (in a manner that considers the intrinsic fault-tolerant mechanisms of the network protocols and services) in addition to realizing a proactive Failure-Prediction and corresponding reactions avoiding future fault activations, **3)** the node level Monitoring Agent - responsible for orchestrating tools and entities for monitoring and anomaly detection on node level as well as among nodes with respect to communicational behavior, **4)** the node level Self-Optimization Agent - responsible for the key functionality of selecting the optimal set of actions initiated by the different control loops (e.g. Fault-Management control loop, Survivability Agent control loop, internal Monitoring Agent control loops) towards optimizing the network/system operation as well as the operation of the FDH framework itself, and last but not least **5)** the Event Dissemination Agent of FDH - taking care of fault/error/failure/alarm and general events sharing across the FDH instances within in a network in order to enable these entities to react to erroneous states in a distributed manner. The interactions among those key components and other supporting components are captured in the form of sequence diagrams describing the flows of the involved control loops and building a framework, in which key algorithms need to be plugged. During the architectural design of the FDH components, special emphasis was put on aspects relating to the involvement of the network operations personnel, which is constituted first by the configuration and provisioning of required operational models (e.g. models for Fault-Isolation, monitoring policies, Fault-Removal/Mitigation policies etc.) and to the escalation - from the FDH perspective - of faulty conditions which are not automatically resolvable to the network administrators, observing and reacting/tweaking the FDH operations as to achieve improved QoS in the long term operation (i.e. achieving an effective "human in the loop" involvment).

After capturing the architectural aspects of FDH including its embedded interaction flows, the current scientific works proceeds with the design of each of the devised components. Thereby, suitable algorithms are selected and new algorithms are researched, in order to realize the two main aspects of FDH, i.e. **1) scalable and efficient self-healing**, and **2) self- optimization with respect to the executed self-healing actions** in order to improve the overall system's operation together with the FDH operations itself. First, the key structures of the Monitoring Agent are investigated including different policy concepts and abstract interfaces for KPI monitoring and Fault-Detection. Subsequently, different approaches to alarm/incident/event dissemination over connectionless packet switched links (e.g. on top of connectionless UDP) are investigated for the goals of FDH. These techniques include flooding, gossiping and repetitive retransmission incorporating mechanisms such as broadcast and multicast. The analyzed techniques are evaluated in a discrete OMNET++ simulation in various Internet alike topologies and links, in order to prove their applicability for the purposes of FDH. Furthermore, techniques for scalable and efficient Fault-Isolation are required as a key part of the Fault-Management agent. The resulting investigation leads to the invention of a new advanced **Markov Chain based technique for Fault-Isolation**, which is one of the key innovations of this thesis. The proposed algorithm turns out to be quite effective based on the conducted case studies and experimental simulations for large network architectures. Moreover, it shows some low overhead with respect to memory consumption during the experiments and is extremely fast, which confirms a number of theoretical calculations regarding the time and space complexity of the proposed Fault-Isolation approach. All these

properties of the proposed Markov Chain based algorithm turn the most difficult, cumbersome and computationally hard part of the self-healing control loop into a fast and scalable calculation allowing the FDH to realize fast self-healing reactions in large scale networks. The outcome of the Fault-Isolation procedure has to be subsequently processed in various ways leading to a final reaction of the FDH framework.The reaction might be given by the reconfiguration of a firewall or the restart of a NIC/ node to give some examples. Therefore, different policy concepts are required which are again presented and evaluated with respect to scalability and evaluation time in the corresponding chapters of this thesis. Such policies are utilized for different tasks including the decision on which Fault- Removal actions to be selected or the assessment (i.e. check) of either the Fault-Isolation or Fault- Removal outcome is valid and successful. Thereby, in case of a negative assessment the overall situation should be escalated to the network operations personnel since obviously the FDH is unable to cope with the challenges to the network.

Having presented the basic self-healing function of the FDH framework, the thesis flow proceeds with describing the self-optimization aspects of the proposed architecture. Self-optimization is proposed at various spots within the introduced node and network level agents' eco-systems: **1)** adaptation of **Monitoring Job Parameters**, **2) Fault-Propagation Model** adaptations, **3) Context-aware Self-healing Actions** based on the history of executed control loops, **4) Selection of Optimal Tentative Actions**, and **5) Avoiding Contradictory Tentative Actions**. Whilst the majority of these aspects would be realized by traditional policy concepts or data mining methods (in case of the Fault-Propagation Model), a main innovation of this thesis is given by **the design/specification, prototyping and evaluation of a dedicated agent (Self- Optimization Agent) for synchronizing the tentative actions coming from multiple parallel running control loops, with the goal of selecting the optimal tentative (agent) activities for execution and avoiding contradictions in parallel control loop reactions, thereby preventing emerging situations where multiple contradicting actions worsen the situation in the network/system instead of resolving an erroneous state**. A Self-Optimization Agent can act on node or network level thereby synchronizing and optimizing the actions of a set of belonging FDH agents. In order to achieve that, a Self- Optimization Agent solves an optimization problem regarding the submitted tentative actions. The current thesis derives the mathematical form of the optimization problem and seeks for different ways to efficiently solve it, given that the mathematical formulation leads to an NP-hard type of problem. Correspondingly, different methods for efficient and scalable (almost) real-time solving of the action synchronization problem are investigated. These methods include the machine learning reformulation of the optimization problem leading to a training process for obtaining key parameters from previously acquired training data (consisted of tentative actions acquired during FDH operations).The resulting computation proves to be fast and scalable with minimal memory consumption overhead, allowing to execute tentative actions' synchronization and optimization for the purpose of efficient and swift distributed self-healing.

To complete the architectural and algorithmic design aspects of the proposed framework, the thesis deals with the involvement of the network operations personnel in the process of self-healing. Hence, the various pre-configurations are described, to be conducted by the administrators tweaking the FDH operations. This includes the provisioning of different models required for the FDH operations, including different policy models - e.g. Fault-Removal policies, different monitoring policies (Admission Policies, Adaptation Policies ...) or Fault-Isolation Assessment policies - as well as a Fault-Propagation Model that is required for (almost) real-time Fault-Isolation in the network devices. In addition, a set of configurations are required for the Self-Optimization agent including the provisioning of a so-called impact matrix which captures the impact of the pre-configured FDH actions (resulting from various policies) on different KPIs relating to the per-

formance of the network/system in question. Based on the provisioning procedures elucidated in this thesis, the admin team is facilitated to provide FDH with all required artifacts for improving the QoS of the managed network and services. On the contrary, in cases when the deployed FDH agents determine that the self- healing processes do not cope with the challenges to the network, they should be able to escalate the current faulty conditions to the network operations personnel and describe the current situation as thoroughly as possible - including various monitored KPIs, deployed model instances, logs of executed control loops and alarm/incident/event descriptions of relevance. Thereby, a set of criteria is required and correspondingly defined enabling the FDH agents to decide on whether a particular situation should be escalated or not. The above elucidated escalation data allows the administrators to take corrective actions, restore network/service functionality, analyze the provided history and improve the FDH operational models, based on some data mining or machine/statistical learning techniques.

After completing the conceptual discussions, the following chapters proceed with the experiments' part of this thesis. This includes the description of the implemented experimental FDH prototype as well as the testbed used for trials and evaluation. Thereby, it should be noted that the testbed originates from a large scale European FP7 project involving several research and industrial partners - including operators and vendors. Subsequently, scalability and overhead investigations are provided which consist of a theoretical complexity analysis and different experiments pushing key input parameters to the extreme, in order to investigate the practical FDH performance based on the developed prototype. The obtained results clearly demonstrate the scalability and effectiveness of the FDH framework thereby inducing minimal overhead in the network devices. Subsequently, a set of case studies are presented which were conducted based on the FDH architecture and prototype and proved the usefulness of the self-healing concepts for different networking problems identified within the aforementioned large scale European FP7 project. The main insights are given by a combined complex case study involving multiple fault-injections over a longer period of time and measuring the FDH performance with respect to the number of automatically removed networking problems. The conducted experiments demonstrate the improvements brought by FDH regarding the processes of network management and control with the goal of providing and improved QoS in distributed network environments.

14.2 Discussion

After the summary of the performed research, the following subsections discuss on how the goals of this thesis - defined in chapter 1 - were achieved, as well as how the conducted scientific work relates in general to the current dynamics of relevant scientific and technological developments. At first, the thesis problem statement from section 1.1.1 is elucidated, followed by relating the obtained results the stated research hypothesis from section 1.1.2. Moreover, the target objectives are put into the picture - these target objectives were defined in section 1.2.1 and section 1.2.2 in the form of expected scientific output and technological objectives regarding the execution of the envisioned thesis research. Finally, the FDH framework and its conceptual and technology background are overally related to current research and development efforts in the domain.

14.2.1 Addressing the Problem Statement

According to the problem statement description in section 1.1.1, this dissertation deals with a couple of challenges the first being **the identification of software components which operate in a**

distributed manner inside the network elements, and increase the self-healing capabilities of a fixed IP network infrastructure. The required software components were identified after a thorough requirements' analysis in chapter 5 based on various inputs including a literature/standards and best practices review of the current situation as well feedback from various vendors and operators - e.g. from a large scale European project but also from industrial projects conducted at the Fraunhofer FOKUS institute during the past years. Thereby, each component was described in detail in chapter 6 and designed to be operating in a distributed manner towards enhancing the self-healing capabilities of a particular fixed IP network infrastructure. Furthermore, the distributed interaction flows among the FDH components across an FDH managed/controlled network are described in detail thereby capturing and illustrating how distributed self-healing with self-optimization features is intended within the current dissertation.

Additional modules of this dissertation's problem statement aim at proving the **feasibility**, **efficiency** and **scalability** of the proposed FDH framework. The feasibility was mainly shown by the prototype described in chapter 10 and deployed in the testbed from chapter 11, in order to conduct various trials and experiments. Thereby, the conducted experiments were based on case studies which were defined based on literature research and feedback from network operators involved in the aforementioned large scale EFIPSANS EU research project. The case studies were used to test the ability of FDH to detect erroneous states and remediate/resolve those in the long term operation of the network. Thereby, it is shown that FDH can **efficiently** improve the QoS and the resilience/robustness of the network in question. In addition, these case studies are the basis for a series of experiments pushing different operational parameters (e.g. network size, number of simultaneous faults, number of actions for self-optimization ...) to the extreme thereby experimentally evaluating the **scalability** and overhead of the implemented prototype. The conducted measurements indicate excellent scalability features of the FDH framework based on the implemented prototype, as part of the feasibility study of the current dissertation. These evaluations match the theoretical results - i.e. complexity analysis - that was performed ahead for the various algorithms and design patterns, which were researched, specified and utilized for the FDH.

14.2.2 Research Hypothesis

The corresponding research hypothesis was elucidated on in section 1.1.2. It basically consists of a set of author's claims relating to the key aspects of the problem statement. Within the first one the author claims that it is possible to **(1) design and specify a software agent based framework for distributed self-healing in fixed IP networks**. This claim was clearly addressed by the requirements analysis chapter 5 and the chapter on the framework design (i.e. chapter 6). The second author's claim states that it is possible to **(2) select or devise efficient and scalable algorithms for the different tasks within the emerging self-healing framework**. In that context, chapter 7 deals with the selection of existing and the definition of new algorithms for the different tasks around the processes of self-healing - including Fault-Detection, Fault-Mitigation, Fault-Isolation and Fault-Removal. The main innovation of chapter 7 is given by the definition of a novel Markov Chain based reasoning scheme for Fault-Isolation, which proves to be scalable and very efficient thereby solving the complex and cumbersome Fault-Isolation problem. Moreover, chapter 7 selects, adapts and modifies different algorithmic schemes for the purposes of FDH, thereby presenting a large range of theoretical and experimental results confirming the claimed scalability. The next module of the research hypothesis is given by the claim that **(3) FDH can be designed in a way that it optimizes its reactions and operational parameters (i.e. self-optimization features), in order to continuously improve the QoS of the network**. This third aspect is addressed on architec-

tural level (see section 6.2.2) as well as on algorithmic framework level in chapter 8. Thereby, the main innovation is given by an advanced scheme for tentative action synchronization, which means that the proposed reactions coming from the multiple control loops are analyzed and only those are allowed to proceed, which are going to provide the best QoS network/system optimization as a combined set of actions. Furthermore, the research hypothesis claims that it is possible to **(4) design the framework in a way, such that the network operations personnel does not lose control over the network/system in question, but rather stays in the loop by monitoring and intervening (if required), in order to resolve problems or improve the operation of FDH**. This fourth aspect has been addressed on architectural design level in chapter 6 and chapter 9, where a special set of interfaces has been devised between the FDH components across the network and the human administrators. Thereby, various required interaction flows were specified which determine either the provisioning of FDH operational models by the network operations personnel, or the escalation of erroneous states, which are not resolvable by FDH means, to the network operations personnel. In this line of thought, chapter 6 defines a set of criteria which the distributed FDH components should use, in order to determine whether a faulty condition should be escalated to the operations personnel. In addition, chapter 6 discusses on technical aspects regarding the realization of the interfaces between the FDH instances and the human experts monitoring, observing and tweaking the self-healing operations. This includes the provisioning of operational models based on different meta-models and ontologies as well as the communication mechanisms for escalation to the employed Network/Element Management System. Having met all the previous conceptual claims, the thesis proceeds with addressing the author's hypothesis that it is possible to **(5) implement the proposed FDH framework for distributed self-healing**. This aspect is addressed by the prototype described in chapter 10 and deployed for the purpose of various experimentations in the testbed described in chapter 11, originating from a large scale FP7 EU project. The prototype components are employed for evaluating the sixth claim of the research hypothesis, which is formulated as **(6) the FDH capability to increase the availability of fixed IP networks**. For this purpose, a number of case studies (chapter 13) were described, illustrating the capabilities of FDH to detect, isolate and remove different network problems. In addition, a complex case study - composed from the different network problems and involving the continuous fault-injection (based on the aforementioned network challenges) of longer series of erroneous states - was used in order to evaluate the FDH reactions in the long term operation of a network. In this case, it was observed that FDH could significantly improve the resilience of the underlying network/system by automatically removing and mitigating faulty conditions, thereby resolving on average 70% of the introduced faults. A number of further experiments (e.g. chapter 7 and chapter 12) relating to the single FDH components (i.e. agents) combined with theoretical complexity analysis (with respect to time and space) were used for proving the next claim of the research hypothesis - **(7) FDH can scale its operations with a growing size of relevant parameters (e.g. number of incidents)**. Thereby, one can clearly observe that all the involved processes and algorithms were designed in a way that they scale with growing network size and incident numbers to be processed simultaneously. During these experiments, the overhead produced by the FDH components was monitored and corresponding results were presented in chapter 7 and chapter 12, thereby proving the last claim of the research hypothesis -namely that **(8) FDH operates efficiently with minimal overhead (e.g. memory consumption and CPU utilization) within the network devices**.

14.2.3 Meeting the Target Objectives

In addition to the above research hypothesis, section 1.2 and section 1.2.1 defined a number of target objectives, which were listed as **thesis' objectives** and **expected scientific output**. Thereby, the objectives are mainly formulated as the means and aims to be achieved, in order to fulfill the research hypothesis as discussed in the previous section. Hence, in many aspects the current manuscript addresses these aspects similarly as it does with the research hypothesis from above.

The formulated target objectives include **the identification of software components, which operate in a distributed manner inside the network nodes, and increase the self-healing capabilities of routers and end systems within a fixed IP network infrastructure** - addressed mainly by chapter 5 (Requirements Analysis), chapter 6 (Framework Specification) and chapter 7 (Realization of the Self-Healing Function). Furthermore, the target objectives encompass the need for the specified components and mechanisms **to proactively and reactively respond to faulty conditions**, which is addressed in the chapters listed for the first point. The goal for the **evaluation of existing mechanisms, and where required, the development of new algorithms for the emerging framework** are handled mainly in chapters 7 and 8, dealing with the selection and definition of efficient mechanisms for the self-healing processes (Fault-Mitigation, Fault-Detection, Fault-Isolation, Fault-Removal ...) and for the self-optimization features of FDH. Thereby, the target objective of **incorporating self-optimization features** is incorporated on multiple levels of the FDH framework starting from the requirements (chapter 5), proceeding to the architectural specification (chapter 6) and rounding up the picture with the development of sophisticated algorithms for tentative action selection and optimization (chapter 8), as well as different evaluations of the effectiveness, scalability and overhead produced by the self-optimization features (chapter 12). This is complemented by **an extensive theoretical and experimental evaluation to prove the effectiveness of the components and algorithms with respect to self-healing, self-optimization as well as scalability and overhead (e.g. CPU utilization or memory consumption) in the network elements**, addressed whilst deriving each algorithm/mechanism/component as well as in the dedicated case study chapter 13 and in the belonging chapter12 on scalability and experimental performance/overhead evaluation. Furthermore, the target of **experimentally showing that the framework as a whole can function and successfully increase the QoS of a network/system** is achieved through a combined case study within section 13.4. Thereby, one can observe that on average 70% of the experimentally introduced fault-injections were resolved by FDH in a longer series of experiments within the testbed of a large scale European FP7 project. This - combined with the single case studies of the same chapter and the various elucidations across the thesis - leads to **scientifically showing that the proposed concepts have the potential to improve the reliability of significant parts of the Internet**. All the conducted experiments, case studies and measurements were performed **based on a prototype that was implemented and utilized to execute the selected case studies of industrial and scientific relevance** - these case studies (chapter 13) were discussed and worked with multiple research and industrial partners from a large European project context, and executed in a large scale testbed (chapter 11) based on the FDH prototype (chapter 10). As an overall conclusion: based on the prototype, case studies and belonging evaluations, it can be claimed that **FDH simplifies traditional Fault-Management processes for the network operations personnel, and correspondingly can help achieving an OPEX (operational expenditure) reduction**, which is also the last target objective in section 1.2.1 that remained unaddressed.

14.2.4 Relation to Modern Trends

The current section summarizes briefly the relation between the presented FDH research concepts and various trends in modern networking and distributed systems.

Autonomic Computing and Networking: Autonomic Computing is a trend that was extremely popular in the past decade leading to a large number of research projects and scientific publications, as well as numerous industry fora and standardization activities. Thereby, different initiatives dealt with refining and re-defining agent based concepts, in order to specify different autonomic entities that are capable of executing feedback control loops (e.g. IBM MAPE) to control their assigned resources. In that context, the FDH entities can be easily mapped to different autonomic entity models (e.g. GANA Decision Elements, Autonomic Computing Elements from the CASCADAS initiative, Autonomic Network Managers from IBM ...) thereby relating the different FDH self-healing and self-optimization processes to the steps within the corresponding autonomic control loops. Hence, the FDH can be easily viewed and understood in the picture/context of Autonomic Computing and Networking.

Cloud Computing: Cloud Computing is a trend that comes with an increased virtualization of networks and services. To a large extent, Cloud Computing simplifies the management of complex distributed infrastructures and increases the robustness and self-healing of those by providing the option to easily restart and migrate virtual machines, as well as to scale automatically the resources (i.e. virtual machines). In that sense, Cloud Computing can be understood as a competition to the ideas of FDH, given that both increase the robustness and self-healing of a distributed system. However, it is indeed possible to combine aspects of Cloud Computing and FDH thereby increasing the effectiveness of both approaches. On one hand, it is possible to deploy FDH in virtual machines and let it act as controller for a particular network/system, thereby ensuring the resilience and robustness of FDH itself. On the other hand, it is possible to utilize FDH in order to manage virtual appliances thereby boosting the effectiveness of the Fault-Removal and Fault-Mitigation processes - e.g. VM restart, VM migration, and elastic resource scaling are techniques which can be easily applied for the purposes of FDH.

SDN/NFV: SDN and NFV have been thoroughly reviewed in section 2.2.9 and constitute the latest advance in research and development with respect to network control and management. Thereby, the basic idea is to separate the data and control planes by placing special SDN/ NFV controller outside the network nodes - in contrast to Autonomic Networking where the autonomic entities are mostly meant to operate within the devices - and allow those controllers to steer and control the nodes over special interfaces, e.g. SDN/OpenFlow based interfaces or hypervisor/container interfaces in relation to NFV. As reviewed in section 2.2.9, different initiatives have followed the path to implement various network functions (e.g. routing) in the form of SDN controllers. Hence, it is possible to view FDH as a controller for a node or a particular network area, thereby incorporating Fault-Mitigation/Removal and Monitoring mechanisms based on SDN/NFV techniques. Hence, FDH instances would be placed in distinct nodes (i.e. not within the forwarding and computing/service providing devices) in the control plane of a network and will increase the robustness of the distributed network infrastructure by implementing self-healing behaviors with device access over the special SDN/NFV interfaces (e.g. OpenFlow) in addition to traditional protocols such as SNMP, NETCONF, TR-69 and CLI.

Modern Network Management: Modern Network Management is largely influenced by the above elucidated trends (i.e. Autonomic Management, Cloud Computing and SDN/NFV). In this line of thought, the previous paragraphs have explained on the relation and possible integration of FDH

into the concepts in question. Hence, the shift towards latest network management and control practices is indeed possible. A vital aspect - when relating and placing FDH in the light of Modern Network Management - is given by the need to increase the level and improve the integration with existing NMS/EMS/OSS systems. A first exemplary integration of FDH to the open source Nagios NMS was demonstrated in chapter 10. However, the research topic has the potential for a larger range of activities including processes and concepts for developing NMS/EMS/OSS-FDH integration components, tools for the provisioning of FDH operational models, analytical tools for understanding the large amounts of FDH operational data and deriving improved operational models, as well the modularity and composability of FDH operational models depending on the various network parameters such as network topology, deployed protocols/service and specific network element implementation aspects. All these aspects are elucidated in the outlook section of this chapter.

Self-Organizing Networks (SON): SON has established itself as part of 3G/4G and potentially 5G mobile communication networks that deals with various self-* aspects including self-healing. The SON aspects were reviewed throughout this thesis, especially in chapter 3 and chapter 4. One of the things that stand out is that SON is very focused on particular scenarios without capturing the various aspects, processes, operational models and required algorithms for efficient self-healing with self-optimization features. Hence, FDH can also be deployed in the context of mobile networks - e.g. in combination with the above elucidated SDN/NFV aspects - and can enhance the capabilities of SON when it comes to the resilience of 3G/4G and potentially 5G networks.

The above discussions point out some of the possible futures research directions in the scope of FDH. Therefore, the following section provides a general outlook on potential research beyond the horizon of this thesis and identifies further activities towards increasing the potential for industrial FDH deployments.

14.3 Outlook

A number of exciting research topics emerge as a result of the (conceptual) groundwork, which was laid down in this thesis. In general, it can be observed that the prototype provides a good starting point for the execution of further case studies in more sophisticated network environments with further network technologies, topologies and services. In addition, the emerged prototype needs further hardening, which will come with the application to additional case studies and the execution of various scenarios involving different fault-injection/attacks in the long term operation of belonging experimental network/system. Thereby, a lot of research and development potential is given by the interface between the network operations personnel and the FDH components across the network. On one hand, this relates to the provisioning of the required operational models for the FDH components. On the other hand, there is a need to research and improve the integration of the belonging FDH interfaces into traditional network monitoring/management systems, thereby increasing the level of "human in the loop" involvement. Regarding the provisioning of FDH operational models: an integrated toolset is required that enables the network administrator to conveniently define the reaction policies (e.g. Fault-Removal policies, monitoring Admission/Adaptation/Notification Policies ...) for the relevant FDH components/processes, the Fault-Propagation Model for the process of Fault-Isolation, as well as the different models required for the action synchronization procedures of the SelfOpt-Agent - i.e. among others the potentially influenced metrics/KPIs, their importance for the system fitness, and the impact of the possible actions on the KPIs. Indeed, one can think of a modular approach to the provisioning of

FDH operational models based on the topology of the network, the employed technologies (and implementations) with their known problems, including the possible fault-propagation chains and belonging reactions. Thereby, different business models have the potential to be developed on top of this modular structure for FDH provisioning relating to various SLAs and expected levels of QoS and end customers' satisfaction.

Regarding the integration of FDH with modern trends in the area of networking and distributed system, a natural development would be constituted by incorporating techniques from the domains of SDN, NFV and cloud computing into the FDH mechanisms. More specifically, FDH can play the role of a self-healing SDN controller utilizing SDN/NFV (e.g. OpenFlow) features, in order to dynamically steer and control the network in question. Thereby, it should be possible to control various types of network nodes and segments, including physical hardware, virtual machines and even Linux/Unix containers, e.g. Docker type of containers. Hence, there is a way to easily shift FDH to the modern research and development technologies. By integrating the SDN/NFV/Hypervisor possibilities to influence the network/system within the pre-defined FDH actions, as well as by placing the FDH components on various controller nodes according to the principles of the corresponding reference model (be it SDN or ETSI NFV).

On top of the above considerations, related to increasing the deployment readiness of FDH, further research can be pursued with respect to the algorithms employed for the various tasks and processes of FDH. For example, different monitoring and analysis techniques can be utilized for the tasks of Fault-Isolation and Failure-Prediction, including Support Vector Machines, Ontologies, Gaussian Processes, and Neural Networks. It is also possible, to explore the utilization of Markov Decision Processes for the purpose of Fault-Removal and Fault-Mitigation with intrinsic self-optimization features, thereby embedding the utilization function of interest within the Markov Decision Process. In addition, the topic of alarm/incident dissemination would require further investigation beyond the scope of the research presented in this thesis. It is possible, to apply and develop special dissemination techniques such as multicast trees, reliable multicast, and further intelligent gossiping and flooding methods, in order to intelligently and reliably distribute alarm/incident messages across an FDH controlled network experiencing a faulty condition. Finally, depending on the success of the alarm/incident dissemination research, the application of FDH to dynamic mobile ad-hoc networks with various degrees of link stability can be examined, in order to extend the scope beyond the fixed IP networks, which were at the heart of the current thesis.

Abbreviations

3G Third generation of wireless mobile telecommunications technology

3GPP Third Generation Partnership Project (3GPP) focusing on specifications in the area of mobile networking technology.

4G Fourth generation of wireless mobile telecommunications technology

4D 4D is a proposed architecture for network management that consists of 4 planes including Data Plane, Discovery Plane, Dissemination Plane and Decision Plane

5G Fifth generation of wireless mobile telecommunications technology

ACK A message within the Transmission Control Protocol (TCP) used to acknowledge the receipt of packets

AFI Autonomic network engineering for the self-managing Future Internet - an ETSI industry specification group

ANA Autonomic Network Architecture, a network architecture that emerged from the corresponding FP6 ANA project

AM Autonomic Manager

ANEMA Autonomic Network Management Architecture to Support Self-configuration and Self-optimization in IP Networks

BGP Border Gateway Protocol

BSI Bundesamt für Sicherheit in der Informationstechnik

BSS Business Support System

CASCADAS FP6 CASCADAS Project - Component-ware for Automatic aware Situations Communications for a Dynamic and Adaptable Service

CAPEX CAPital EXpenditure

CIM Common Information Model

CIM-SPL CIM Simplified Policy Language

CLI Command Line Interface

CONMan Complexity Oblivious Network Management is another proposed architecture for network management which was widely discussed within the research community

COPS Common Open Policy Service - a protocol for distributing and executing policies across a networked system

API Application Programming Interface

DE Decision Element - a key component of the GANA architecture

Den-ng Next generation of the Directory Enabled Networks model aiming at improved policy based network management

CMIP Common Management Information Protocol

DHCP Dynamic Host Configuration Protocol

DHCPv6 Dynamic Host Configuration Protocol version 6

DMTF Distributed Management Task Force - an independent standardization body

DNS Domain Name System

DOM Document Object Model - an XML parsing and handling paradigm

DSL Digital Subscriber Line

ECA Event-Condition-Action

EFIPSANS FP7 EFIPSANS project - Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services

ENISA European Network and Information Security Agency

EMF Eclipse Modeling Framework

EMS Element Management System

EPC Evolved Packet Core - the core network architecture of the LTE mobile communications standard

ETSI European Telecommunications Standards Institute

FCAPS FCAPS is an acronym for Fault, Configuration, Accounting, Performance, Security relating to TMN

FDDI Fiber Distributed Data Interface

FDH Framework for Distributed Self-Healing

FOCALE An Architecture for Autonomic Networking which was proposed and widely discussed in the research community

FR Functional Requirements

GANA Generic Autonomic Networking Architecture

GB Gigabyte

GLPK GNU Linear Programming Kit

GME Generic Modeling Environment

(G)MPLS (Generalized) Multi-Protocol Label Switching - protocol for routing, QoS and virtual private networking based on label switching

HTTP Hypertext Transfer Protocol

IANA Internet Assigned Numbers Authority

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

ICMP Internet Control Message Protocol

ICMPv6 Internet Control Message Protocol version 6

IDS Intrusion Detection System

IGMP Internet Group Management Protocol - an IPv4 based protocol for enabling multicast

ISO International Organisation for Standardisation

IoT Internet of Things

IP Internet Protocol

IPFix Internet Protocol Flow Information Export - an architecture/protocol for monitoring IP networks

IPS Intrusion Prevention System

ITU International Telecommunication Union

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

ISP Internet Service Provider

ICT Information and Communication Technology

IT Information Technology

KB Kilobyte

KPI Key Performance Indicator

LAN Local Area Network

LTE Long Term Evolution - a standard for high-speed mobile communication networks

M2M Machine-2-Machine communication

MAPE Monitor-Analyse-Plan-Execute - the phases of the autonomic control loop proposed by IBM

MB Megabyte

MIB Management Information Base - a key information base of the SNMP network monitoring protocol

MLD Multicast Listener Discovery - an IPv6 based protocol for establishing multicast trees over IPv6 networks

MSU Mobile Sensor Unit

MTTR Mean Time To Repair

NETCONF Network Configuration Protocol

NFR Non-functional Requirements

NMS Network Management System

OSI Open Systems Interconnection (OSI)

OSPF Open Shortest Path First - a widely deployed IP routing protocol in today's networks

OPEX Operational expenditure

OCSP Online Certificate Status Protocol

OWL Web Ontology Language

OSS Operation Support Systems

P2P Peer-to-Peer Networking

PCAP Packet Capture library

PDP Policy Decision Point - a concept within the COPS protocol for policy distribution and management

PEP Policy Enforcement Point - a concept within the COPS protocol for policy distribution and management

PGM Pragmatic Generic Multicast - a reliable multicast protocol

PMTU Path Maximum Transmission Unit

QoE Quality of Experience

QoS Quality of Service

RAN Radio Access Network

RAT Radio Access Technology

Req Requirement

RDF Resource Description Framework

RFC Request for Comments

RIP-ng Routing Information Protocol - next generation

RMON Remote Network MONitoring - a network monitoring protocol specified at IETF

RTT Round Trip Time

SOA Service Oriented Architecture

SOAP Simple Object Access Protocol

SDN Software Defined Networking

SDH Synchronous Digital Hierarchy is a standardized protocol for data transmission over optical fiber

SDR Software Defined Radio

SID Shared Information and Data model of TMF

SNMP Simple Network Management Protocol

SONET Synchronous optical networking is another standardized physical layer protocol for transmitting data over optical fiber links

TCP Transmission Control Protocol

TLV Type-Length-Value or Tag-Length-Value

TMN Telecommunications Management Network

TMF TeleManagement Forum

TR-69 Technical Report 069 - a technical specification of the Broadband Forum defining a protocol for remote management of devices at the customer-premises

NIC Network Interface Card

NFV Network Functions Virtualization

VNF Virtualized Network Function

UDP User Datagram Protocol

UML Unified Modeling Language

UMTS Universal Mobile Telecommunications System - third generation mobile cellular system for mobile networks.

UTRAN Universal Terrestrial Radio Access Network - an access mobile communication network corresponding to the UMTS standard

VoIP Voice over Internet Protocol

VM Virtual Machine

VPN Virtual Private Network

WiMAX Worldwide Interoperability for Microwave Access - a family of IEEE based wireless communication standards

WLAN Wireless Local Area Network

XACML eXtensible Access Control Markup Language

XML Extensible Markup Language

XORP eXtensible Open Router Platform

Bibliography

[3GP13] ETSI TS 132.622, Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Telecommunication management; Configuration Management (CM); Generic network resources Integration Reference Point (IRP); Network Resource Model (NRM) (3GPP TS 32.622 version 11.1.0 Release 11), V11.1.0. http://www.etsi.org/deliver/etsi_ts/132600_132699/132622/11.01.00_60/ts_132622v110100p.pdf, July 2013.

[3GP16a] 3GPP SON: Self-Organizing Networks. http://www.3gpp.org/technologies/keywords-acronyms/105-son, 2016.

[3GP16b] ETSI TS 32.541, Telecommunication management - Self-Organizing Networks(SON) - Self-healing concepts and requirements, Release 13. http://www.etsi.org/deliver/etsi_ts/132500_132599/132541/13.00.00_60/ts_132541v130000p.pdf, Feb 2016.

[4WA17] The FP7 4WARD Project. http://www.4ward-project.eu/, 2017.

[AAFR10] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri. An Adaptive Sampling Algorithm for Effective Energy Management in Wireless Sensor Networks With Energy-Hungry Sensors. *IEEE Transactions on Instrumentation and Measurement*, 59(2):335–344, Feb 2010.

[ACL17] Configuring IP Access Lists. http://www.cisco.com/c/en/us/support/docs/security/ios-firewall/23602-confaccesslists.pdf, 2017.

[AFRR+10] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[AHCBCD06] Diala Abi Haidar, Nora Cuppens-Boulahia, Frederic Cuppens, and Herve Debar. An Extended RBAC Profile of XACML. In *Proceedings of the 3rd ACM Workshop on Secure Web Services*, SWS '06, pages 13–22, New York, NY, USA, 2006. ACM.

[AJ94] B. Albert and A.P. Jayasumana. *FDDI and FDDI-II: Architecture, Protocols, and Performance*. Artech House telecommunications library. Artech House, 1994.

[AL10] Anwesh Adhikari and Latif Ladid. INFSO-ICT-215549-WP6-D6.6: D6.6 Socio-economic Studies Report-2. Jan 2010.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004.

[AMB02] Toufik Ahmed, Ahmed Mehaoua, and Raouf Boutaba. Dynamic QoS Adaptation Using COPS and Network Monitoring Feedback. In *Management of Multimedia on the Internet, 5th IFIP/IEEE International Conference on Management of Multimedia Networks and Services, MMNS 2002, Santa Barbara, CA, USA, October 6-9, 2002, Proceedings*, pages 250–262, 2002.

[AMBK+07] D. Abusch-Magder, P. Bosch, T. E. Klein, P. A. Polakos, L. G. Samuel, and H. Viswanathan. 911-NOW: A network on wheels for emergency response and disaster recovery operations. *Bell Labs Technical Journal*, 11(4):113–133, Winter 2007.

[ANA16] Autonomic Network Architecture (ANA). http://www.ana-project.org/, 2016.

[AO16] Dhaminda B. Abeywickrama and Eila Ovaska. A survey of autonomic computing methods in digital service ecosystems. *Service Oriented Computing and Applications*, pages 1–31, 2016.

[APA17] Apache Jena: A free and open source Java framework for building Semantic Web and Linked Data applications. https://jena.apache.org, Jan 2017.

[Aut03] Achim Autenrieth. *Differentiated resilience in IP based multilayer transport networks*. PhD thesis, TU München, 2003.

[Aut17a] AutoI: Autonomic Internet. http://www.autoi.ics.ece.upatras.gr/autoi/index.php, 2017.

[AUT17b] AUTOSAR: AUTomotive Open System ARchitecture. http://www.autosar.org/, June 2017.

[Bar08] Wolfgang Barth. *Nagios: System and Network Monitoring*. No Starch Press, San Francisco, CA, USA, 2nd edition, 2008.

[BBM98] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, Jan 1998.

[BBPK13] Sandro Bimonte, Kamal Boulil, François Pinet, and Myoung-Ah Kang. Design of Complex Spatio-multidimensional Models with the ICSOLAP UML Profile - An Implementation in MagicDraw. In *ICEIS 2013 - Proceedings of the 15th International Conference on Enterprise Information Systems, Volume 1, Angers, France, 4-7 July, 2013*, pages 310–315, 2013.

[BCE+16] J. Barron, M. Crotty, E. Elahi, R. Riggio, D. R. Lopez, and M. P. de Leon. Towards self-adaptive network management for a recursive network architecture. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1143–1148, April 2016.

[BCF+17] A.F. Benner, C.J. Colonna, J.R. Flanagan, D.F. Riedy, and H.M. Yudenfriend. Isolating the sources of faults/potential faults within computing networks, August 1 2017. US Patent 9,722,863.

[BCX+11] Michael H. Behringer, Ranganai Chaparadza, Li Xin, Heikki Mahkonen, and Razvan Petre. IP based Generic Control Protocol (IGCP). Internet-Draft draft-chaparadza-intarea-igcp-00, Internet Engineering Task Force, July 2011. Work in Progress.

[BDJS10] Jason Barron, Steven Davy, Brendan Jennings, and John Strassner. A Policy Authoring Process and DEN-ng Model Extension for Federation Governance. In *Modelling Autonomic Communication Environments - 5th IEEE International Workshop, MACE 2010, Niagara Falls, Canada, October 28, 2010. Proceedings*, pages 73–86, 2010.

[BDK+12] Craig Boutilier, Rajarshi Das, Jeffrey O. Kephart, Gerald Tesauro, and William E. Walsh. Cooperative Negotiation in Autonomic Systems using Incremental Utility Elicitation. *CoRR*, abs/1212.2443, 2012.

[BDmB06] Javier Baliosian, Ann Devitt, and Anne marie Bosneag. The Omega Architecture: Towards Adaptable, Self-Managed Networks. In *in proceedings of the 1st Annual Workshop on Distributed Autonomous Network Management Systems*, 2006.

[BDRG17] Soukaina Bouzghiba, Hamza Dahmouni, Anouar Rachdi, and Jean-Marie Garcia. *Towards an Autonomic Approach for Software Defined Networks: An Overview*, pages 149–161. Springer Singapore, Singapore, 2017.

[BE02] David Braginsky and Deborah Estrin. Rumor Routing Algorthim for Sensor Networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pages 22–31, New York, NY, USA, 2002. ACM.

[Bea16] BeanShell: Lightweight Scripting for Java. http://www.beanshell.org, dec 2016.

[Bea17] The Beacon SDN Controller. https://openflow.stanford.edu/display/Beacon/Home, Apr 2017.

[Bel04] Donald Bell. The class diagram: An introduction to structure diagrams in UML 2. https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/, September 2004.

[Ber93] M.A. Berger. *An introduction to probability and stochastic processes*. Springer texts in statistics. Springer-Verlag, 1993.

[BF06] Hitesh Ballani and Paul Francis. Conman: Taking the complexity out of network management. In *Proceedings of the 2006 SIGCOMM Workshop on Internet Network Management*, INM '06, pages 41–46, New York, NY, USA, 2006. ACM.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BJP05] A. Beller, E. Jamhour, and M. C. Penna. Dynamic DiffServ configuration using COPS-PR. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)*, pages 8 pp.–467, Nov 2005.

[BJT+10] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May. The autonomic network architecture (ANA). *IEEE Journal on Selected Areas in Communications*, 28(1):4–14, January 2010.

[Bla94] Uyless D. Black. *Network Management Standards: SNMP, CMIP, TMN, MIBs and Object Libraries*. McGraw-Hill, Inc., New York, NY, USA, 2nd edition, 1994.

[BM08] E. G. Birgin and J. M. Martínez. Improving Ultimate Convergence of an Augmented Lagrangian Method. *Optimization Methods Software*, 23(2):177–195, April 2008.

[BMQS08] J. Baliosian, K. Matusikova, K. Quinn, and R. Stadler. Policy-based self-healing for radio access networks. In *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, pages 1007–1010, April 2008.

[Bou03] Jim Bound. Dynamic Host Configuration Protocol for IPv6 (DHCPv6). RFC 3315, July 2003.

[BP98] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Seventh International World-Wide Web Conference (WWW 1998)*, 1998.

[BPW98] A. Bieszczad, B. Pagurek, and T. White. Mobile agents for network management. *IEEE Communications Surveys*, 1(1):2–9, First 1998.

[Bra89] H-W. Braun. Models of Policy Based Routing. RFC 1104, June 1989.

[BRI16] BRITE: Boston University Representative Internet Topology Generator. https://www.cs.bu.edu/brite/, 2016.

[BSB+16] Sreram Balasubramaniyan, Seshadhri Srinivasan, Furio Buonopane, B. Subathra, Jüri Vain, and Srini Ramaswamy. Design and verification of cyber-physical systems using truetime, evolutionary optimization and {UPPAAL}. *Microprocessors and Microsystems*, 42:37 – 48, 2016.

[BSI17] Bundesamt für Sicherheit in der Informationstechnik. https://www.bsi.bund.de, 2017.

[BTadG+05] Andreas Binzenhöfer, Kurt Tutschku, Björn auf dem Graben, Markus Fiedler, and Patrik Arlos. A P2P-Based Framework for Distributed Network Management. In *Wireless Systems and Network Architectures in Next Generation Internet, Second International Workshop of the EURO-NGI Network of Excellence, Villa Vigoni, Italy, July 13-15, 2005, Revised Selected Papers*, pages 198–210, 2005.

[But97] David R. Butenhof. *Programming with POSIX Threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[BWA16] Oren Barkan, Jonathan Weill, and Amir Averbuch. Gaussian Process Regression for Out-of-Sample Extension. *CoRR*, abs/1603.02194, 2016.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495, 5946.

[Cai12] Zheng Cai. *Maestro: Achieving Scalability and Coordination in Centralizaed Network Control Plane*. PhD thesis, Houston, TX, USA, 2012. AAI3521292.

[CAS16] The CASCADAS project. http://acetoolkit.sourceforge.net/cascadas/, Dec 2016.

[CDF+02] Bradley Cain, Dr. Steve E. Deering, Bill Fenner, Isidor Kouvelas, and Ajit Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, October 2002.

[CDL+10] Zhen Chen, Fa-Chao Deng, An-An Luo, Xin Jiang, Guo-Dong Li, Run hua Zhang, and Chuang Lin. Application level network access control system based on TNC architecture for enterprise network. In *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*, pages 667–671, June 2010.

[CdLL+17] J. Cámara, R. de Lemos, N. Laranjeiro, R. Ventura, and M. Vieira. Robustness-Driven Resilience Evaluation of Self-Adaptive Software Systems. *IEEE Transactions on Dependable and Secure Computing*, 14(1):50–64, Jan 2017.

[CFP+07] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking Control of the Enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, August 2007.

[Cha08] Ranganai Chaparadza. Requirements for a Generic Autonomic Network Architecture (GANA), suitable for standardizable Autonomic Behavior Specifications of Decision-making-elements (DMEs) for diverse Networking Environments. *IEC Annual review of communications*, 61, 2008.

[Cha09] Ranganai Chaparadza. UniFAFF: a unified framework for implementing autonomic fault management and failure detection for self-managing networks. *Int. Journal of Network Management*, 19(4):271–290, 2009.

[CIM09] CIM Simplified Policy Language (CIM-SPL). http://www.dmtf.org/sites/default/files/standards/documents/DSP0231_1.0.0.pdf, Jul 2009.

[CIM16] DMTF Common Information Model. http://www.dmtf.org/standards/cim, 2016.

[cL08] Jean claude Laprie. From dependability to resilience. In *In 38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, 2008.

[Cla04] Benoit Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.

[Cla08] Benoit Claise. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, January 2008.

[CLCG15] C. Cordray, D. Link, R. Chart, and K. Ginter. Self configuring network management system, July 2015. US Patent 9,077,611.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Dijkstra's algorithm. In *Introduction to Algorithms 2nd edition*, chapter 24, pages 595–599. MIT Press, Sep 2001.

[CMS+14] R. Chaparadza, T. Ben Menem, J. Strassner, B. Radier, S. Soulhi, J. Ding, and Z. Yan. Industry harmonization for unified standards on autonomic management amp; control (amc) of networks and services, sdn and nfv. In *2014 IEEE Globecom Workshops (GC Wkshps)*, pages 155–160, Dec 2014.

[COI17] Coin-OR. http://www.coin-or.org/, Jan 2017.

[CPRW03] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. A Knowledge Plane for the Internet. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 3–10, New York, NY, USA, 2003. ACM.

[CSI+14] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, R. Natella, J. Fan, and W. Ping. Network Function Virtualization: Challenges and Directions for Reliability Assurance. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 37–42, Nov 2014.

[CTK10] Ranganai Chaparadza, Nikolay Tcholtchev, and Vassilios Kaldanis. How Autonomic Fault-Management Can Address Current Challenges in Fault-Management Faced in IT and Telecommunication Networks. In *Access Networks - 5th International ICST Conference on Access Networks, AccessNets 2010 and First ICST International Workshop on Autonomic Networking and Self-Management in Access Networks, SELFMAGICNETS 2010, Budapest, Hungary, November 3-5, 2010, Revised Selected Papers*, pages 253–268, 2010.

[CTS08] Ranganai Chaparadza, Nikolay Tcholtchev, and Ina Schieferdecker. Implementation of the UniFAFF framework for autonomic fault-management in ANA networks. In *Third International Conference on the Latest Advances in Networks, ICLAN 2008. Proceedings : Toulouse, France, December10-12, 2008*, pages 75–81, 2008.

[CV04] Luis Costa and Rolland Vida. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810, June 2004.

[CVP+12] Marco Canini, Daniele Venzano, Peter Perešíni, Dejan Kostić, and Jennifer Rexford. A NICE Way to Test Openflow Applications. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[CWM+13] Ranganai Chaparadza, Michal Wódczak, Tayeb Ben Meriem, Paolo De Lutiis, Nikolay Tcholtchev, and Laurent Ciavaglia. Standardization of resilience & survivability, and autonomic fault-management, in evolving and future networks: An ongoing initiative recently launched in ETSI. In *9th International Conference on*

the Design of Reliable Communication Networks, DRCN 2013, Budapest, Hungary, March 4-7, 2013, pages 331–341, 2013.

[CXG17] Mingjie Cai, Zhengrong Xiang, and Jian Guo. Adaptive finite-time control for a class of switched nonlinear systems using multiple Lyapunov functions. *International Journal of Systems Science*, 48(2):324–336, 2017.

[DAS09] Hajer Derbel, Nazim Agoulmine, and Mikaël Salaün. ANEMA: Autonomic Network Management Architecture to Support Self-configuration and Self-optimization in IP Networks. *Comput. Netw.*, 53(3):418–430, February 2009.

[DDF+06] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, December 2006.

[DDLS01] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder Policy Specification Language. In *Lecture Notes in Computer Science*, pages 18–38. Springer-Verlag, 2001.

[DDV+11] Panagiotis Demestichas, Sudhir S. Dixit, Martin Vigoureux, Mikhail Smirnov, and Antonio Manzalini. Managing an Autonomic Future Internet. *IEEE Network*, 25(6):4–6, 2011.

[Del08] S-Cube Deliverable #CD-JRA-1.3.2:Quality Reference Model for SBA. http://www.s-cube-network.eu/results/deliverables/wp-jra-1.3/Reference_Model_for_SBA.pdf, March 2008.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[DJS07] Steven Davy, Brendan Jennings, and John Strassner. The Policy Continuum – A Formal Model. In *2nd IEEE International Workshop, MACE 2007*, page 65–78, Oct 2007.

[DKB11] P. Dely, A. Kassler, and N. Bayer. OpenFlow for Wireless Mesh Networks. In *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6, July 2011.

[DKdLS08] Yuri Demchenko, Oscar Koeroo, Cees de Laat, and Hakon Sagehaug. Extending XACML Authorisation Model to Support Policy Obligations Handling in Distributed Application. In *Proceedings of the 6th International Workshop on Middleware for Grid Computing*, MGC '08, pages 5:1–5:6, New York, NY, USA, 2008. ACM.

[DKS07] E. Davies, S. Krishnan, and P. Savola. IPv6 Transition/Co-existence Security Considerations. RFC 4942 (Informational), September 2007.

[DL13] C. X. Dou and B. Liu. Multi-Agent Based Hierarchical Hybrid Control for Smart Microgrid. *IEEE Transactions on Smart Grid*, 4(2):771–778, June 2013.

[DLC03] Timon C. Du, Eldon Y. Li, and An-Pin Chang. Mobile Agents in Distributed Network Management. *Commun. ACM*, 46(7):127–132, July 2003.

[DMA16] D-MASON: Distributed Multi-Agents Based Simulations toolkit. https://sites.google.com/site/distributedmason/, 2016.

[DMT17] DMTF: Distributed Management Task Force. https://www.dmtf.org/, 2017.

[DPD17] DPDK: Data Plane Development Kit. http://dpdk.org/, June 2017.

[DPF+08] Willie Donnelly, Manish Parashar, Joel Fleck, David Lewis, and John Strassner. The Role of Standardization in Future Autonomic Communication Systems. *2008 5th IEEE Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2008)*, 00:165–173, 2008.

[Dro97] Ralph Droms. Dynamic Host Configuration Protocol. RFC 2131, March 1997.

[dSBC+10] C.R.P. dos Santos, R.S. Bezerra, J.M. Ceron, L.Z. Granville, and L.M. Rockenbach Tarouco. On using mashups for composing network management applications. *Communications Magazine, IEEE*, 48(12):112 –122, december 2010.

[DYHG17] C. Dou, D. Yue, Q. L. Han, and J. M. Guerrero. Multi-agent system-based event-triggered hybrid control scheme for energy internet. *IEEE Access*, 5:3263–3272, 2017.

[ea06] E.Höfig et. al. On Concepts for Autonomic Communication Elements. In *1st IEEE International Workshop on Modelling Autonomic Communication Environments*, 2006.

[EBBS11] Rob Enns, Martin Bjorklund, Andy Bierman, and Jürgen Schönwälder. Network Configuration Protocol (NETCONF). RFC 6241, June 2011.

[EBG+15] W. Elghazel, J. Bahi, C. Guyeux, M. Hakem, K. Medjaher, and N. Zerhouni. Dependability of Wireless Sensor Networks for Industrial Prognostics and Health Management. *Comput. Ind.*, 68(C):1–15, April 2015.

[EdAG15] J. Espinosa, D. d. Andrés, and P. Gil. Increasing the Dependability of VLSI Systems through Early Detection of Fugacious Faults. In *2015 11th European Dependable Computing Conference (EDCC)*, pages 190–197, Sept 2015.

[efi16a] Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services (EFIPSANS). http://cordis.europa.eu/project/rcn/85542_de.html, 2016.

[efi16b] Exposing the Features in IP version Six protocols that can be exploited/extended for the purposes of designing/building Autonomic Networks and Services (EFIPSANS). https://www.fokus.fraunhofer.de/en/sqc/projects/efipsans, 2016.

[ENI17] European Union Agency for Network and Information Security. https://www.enisa.europa.eu/, 2017.

[ent12] Enterprise Java XACML Implementation. https://code.google.com/archive/p/enterprise-java-xacml/, 2008-2012.

[Eri13] David Erickson. The Beacon Openflow Controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, New York, NY, USA, 2013. ACM.

[ES01] Kjeld Borch Egevang and Pyda Srisuresh. Traditional IP Network Address Translator (Traditional NAT). RFC 3022, January 2001.

[ETH05] Man page: ethtool - Display or change ethernet card settings. http://linuxcommand.org/man_pages/ethtool8.html, March 2005.

[eTO17] TMF: Business Process Framework (eTOM). https://www.tmforum.org/business-process-framework/, 2017.

[ETS13] ETSI GS AFI 002 V1.1.1: Autonomic network engineering for the self-managing Future Internet (AFI); Generic Autonomic Network Architecture (An Architectural Reference Model for Autonomic Networking, Cognitive Networking and Self-Management) . http://www.etsi.org/deliver/etsi_gs/AFI/001_099/002/01.01.01_60/gs_afi002v010101p.pdf, Apr 2013.

[Fal03] Kevin Fall. A Delay-tolerant Network Architecture for Challenged Internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[FFR+11] Mauro Femminella, Roberto Francescangeli, Gianluca Reali, Jae Woo Lee, and Henning Schulzrinne. An enabling Platform for Autonomic Management of the Future Internet. *IEEE Network*, 25(6):24–32, 2011.

[FG97] Stan Franklin and Art Graesser. Is It an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, ECAI '96, pages 21–35, London, UK, UK, 1997. Springer-Verlag.

[FGR+07] Roy Friedman, Daniela Gavidia, Luis Rodrigues, Aline Carneiro Viana, and Spyros Voulgaris. Gossiping on MANETs: The Beauty and the Beast. *SIGOPS Oper. Syst. Rev.*, 41(5):67–74, October 2007.

[FGR+13] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison. Languages for software-defined networks. *IEEE Communications Magazine*, 51(2):128–134, February 2013.

[FHH14] F. Fittkau, A. V. Hoorn, and W. Hasselbring. Towards a Dependability Control Center for Large Software Landscapes (Short Paper). In *2014 Tenth European Dependable Computing Conference*, pages 58–61, May 2014.

[FKM+16] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. *Probabilistic NetKAT*, pages 282–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[FLA13] FLAME: Flexible Large-scale Agent Modeling Environment. http://flame.ac.uk/, 2013.

[FLL+97] David Fisher, Richard Linger, Howard Lipson, Thomas Longstaff, Nancy Mead, and Robert Ellison. Survivable Network Systems: An Emerging Discipline. Technical Report CMU/SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1997.

[Flo62] Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

[Flo17] Floodlight: Open SDN Controller. http://www.projectfloodlight.org/floodlight/, Apr 2017.

[FMA+15] Antonio Filieri, Martina Maggio, Konstantinos Angelopoulos, Nicolás D'Ippolito, Ilias Gerostathopoulos, Andreas Berndt Hempel, Henry Hoffmann, Pooyan Jamshidi, Evangelia Kalyvianaki, Cristian Klein, Filip Krikava, Sasa Misailovic, Alessandro Vittorio Papadopoulos, Suprio Ray, Amir M. Sharifloo, Stepan Shevtsov, Mateusz Ujma, and Thomas Vogel. Software Engineering Meets Control Theory. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15, pages 71–82, Piscataway, NJ, USA, 2015. IEEE Press.

[Fre04] Arnaud Freville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1 – 21, 2004.

[Fre17] The Frenetic project: a family of network programming languages. http://frenetic-lang.org/, Apr 2017.

[GFN15] Iñaki Garitano, Seraj Fayyad, and Josef Noll. Multi-Metrics Approach for Security, Privacy and Dependability in Embedded Systems. *Wirel. Pers. Commun.*, 81(4):1359–1376, April 2015.

[GG67] P. Gilmore and R. Gomory. The theory and computation of knapsack functions. *Operations Research*, 15:1045–1075, 1967.

[GHM+05] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A Clean Slate 4D Approach to Network Control and Management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, October 2005.

[GK94] Michael R. Genesereth and Steven P. Ketchpel. Software Agents. *Commun. ACM*, 37(7):48–ff., July 1994.

[GKP+08] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[GKT13] C. Giuffrida, A. Kuijsten, and A. S. Tanenbaum. EDFI: A Dependable Fault Injection Tool for Dependability Benchmarking Experiments. In *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, pages 31–40, Dec 2013.

[GL16] Mihalis Giannakis and Michalis Louis. A multi-agent based system with big data processing for enhanced supply chain agility. *Journal of Enterprise Information Management*, 29(5):706–727, 2016.

[GLD+14] Xiongzi Ge, Yi Liu, David H.C. Du, Liang Zhang, Hongguang Guan, Jian Chen, Yuping Zhao, and Xinyu Hu. OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack. *SIGCOMM Comput. Commun. Rev.*, 44(4):353–354, August 2014.

[GLP16] GLPK: GNU Linear Programming Kit. www.etsi.org, Dec 2016.

[GME16] GME: Generic Modeling Environment. http://www.isis.vanderbilt.edu/projects/gme/, Dec 2016.

[GPP12] Jochen Gorski, Luís Paquete, and Fábio Pedrosa. Greedy Algorithms for a Class of Knapsack Problems with Binary Weights. *Comput. Oper. Res.*, 39(3):498–511, March 2012.

[Gre95] C. J. Green. Protocols for a self-healing network. In *Military Communications Conference, 1995. MILCOM '95, Conference Record, IEEE*, volume 1, pages 252–256 vol.1, Nov 1995.

[Gro] ITU-T Reference Models-Study Group. I.322: EN-Generic Protocol Reference Model for Telecommunication Networks - Series I: Integrated Services Digital Network Overall Network Aspects and Functions.

[GVCR17] Marisol García-Valls, António Casimiro, and Hans P. Reiser. A Few Open Problems and Solutions for Software Technologies for Dependable Distributed Systems. *J. Syst. Archit.*, 73(C):1–5, February 2017.

[GvSO01] Mariana Gerber, Rossouw von Solms, and Paul L. Overbeek. Formalizing information security requirements. *Inf. Manag. Comput. Security*, 9(1):32–37, 2001.

[Hal11] Edward Haletky. *VMware ESX and ESXi in the Enterprise: Planning Deployment of Virtualization Servers*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2nd edition, 2011.

[HBCS11] Michael Hanrahan, Sarah Banks, Andrea Colmegna, and Tim Spets. *TR-069:CPE WAN Management Protocol, Issue: 1 Amendment 4, Protocol Version: 1.3*. Broadband Forum, July 2011.

[HBZ+14] Jörg Henkel, Lars Bauer, Hongyan Zhang, Semeen Rehman, and Muhammad Shafique. Multi-Layer Dependability: From Microarchitecture to Application Level. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 47:1–47:6, New York, NY, USA, 2014. ACM.

[HCG01] Edwin A. Hernandez, Matthew C. Chidester, and Alan D. George. Adaptive Sampling for Network Management. *J. Netw. Syst. Manage.*, 9(4):409–434, December 2001.

[HDF16] S. Huang, Z. Deng, and S. Fu. Quantifying entity criticality for fault impact analysis and dependability enhancement in software-defined networks. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, Dec 2016.

[HDPT04] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[Her88] R. Herken, editor. *A Half-century Survey on The Universal Turing Machine*, New York, NY, USA, 1988. Oxford University Press, Inc.

[Her00] Shai Herzog. The COPS (Common Open Policy Service) Protocol. RFC 2748, January 2000.

[HFK+14] Vincent C. Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. *NIST Special Publication 800-162: Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. NIST/SEMATECH, 2014.

[HGJL15] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, Feb 2015.

[HGK16] S. Hasbe, H. Gupta, and M. Kashelikar. Using a network bubble across multiple hosts on a disaster recovery site for fire drill testing of a multi-tiered application, may 2016. US Patent 9,336,103.

[HHL06] Zygmunt J. Haas, Joseph Y. Halpern, and Li Li. Gossip-based Ad Hoc Routing. *IEEE/ACM Trans. Netw.*, 14(3):479–491, June 2006.

[HHM04] E. N. Herness, R. J. High, and J. R. McGee. WebSphere Application Server: A foundation for on demand computing. *IBM Systems Journal*, 43(2):213–237, 2004.

[Hib17] Hibernate. http://hibernate.org/, Jan 2017.

[HLDL15] M. Hoffmann, F. Lukas, C. Dietrich, and D. Lohmann. dOSEK: the design and implementation of a dependability-oriented static embedded kernel. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 259–270, April 2015.

[HMR+15] Jong Hun Han, Prashanth Mundkur, Charalampos Rotsos, Gianni Antichi, Nirav H. Dave, Andrew William Moore, and Peter G. Neumann. Blueswitch: Enabling Provably Consistent Configuration of Network Switches. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, pages 17–27, Washington, DC, USA, 2015. IEEE Computer Society.

[HPD+15] Evangelos Haleplidis, Kostas Pentikousis, Spyros Denazis, Jamal Hadi Salim, David Meyer, and Odysseas Koufopavlou. Software-Defined Networking (SDN): Layers and Architecture Terminology. RFC 7426, January 2015.

[HRM+01] Shai Herzog, Francis Reichmeyer, Keith McCloghrie, John D. Seligson, Kwok Ho Chan, Silvano Gai, Andrew H. Smith, and David Durham. COPS Usage for Policy Provisioning (COPS-PR). RFC 3084, March 2001.

[HRW09] C. Hein, T. Ritter, and M. Wagner. Model-driven tool integration with modelbus. In *Workshop Future Trends of Model-Driven Development*, pages 50–52, 2009.

[HRW15] J. Hwang, K. K. Ramakrishnan, and T. Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, 12(1):34–47, March 2015.

[HS15] Mathes M. Heinzl S. *Middleware in Java: Leitfaden zum Entwurf verteilter Anwendungen - Implementierung von verteilten Systemen über JMS - Verteilte Objekte über RMI und CORBA*. Springer-Verlag, 2015.

[hSJK16] Jae ho Shin, Gyoung-Don Joo, and Chulyun Kim. XPath based crawling method with crowdsourcing for targeted online market places. In *2016 International Conference on Big Data and Smart Computing (BigComp)*, pages 395–397, Jan 2016.

[HSV99] Masum Hasan, Binay Sugla, and Ramesh Viswanathan. A Conceptual Framework for Network Management Event Correlation and Filtering Systems. In Morris Sloman, Subrata Mazumdar, and Emil C. Lupu, editors, *1999 IEEE/IFIP International Symposium on Integrated Network Management, IM 1999, Boston, USA, May 24-28, 1999. Proceedings*, pages 233–246. IEEE, 1999.

[HTC+16] Zhenhua Han, Haisheng Tan, Guihai Chen, Rui Wang, Yifan Chen, and Francis C. M. Lau. Dynamic Virtual Machine Management via Approximate Markov Decision Process. *CoRR*, abs/1602.00097, 2016.

[HZZ+10] Honglin Hu, Jian Zhang, Xiaoying Zheng, Yang Yang, and Ping Wu. Self-configuration and Self-optimization for LTE Networks. *Comm. Mag.*, 48(2):94–100, February 2010.

[ibm05] An Architectural Blueprint for Autonomic Computing. Technical report, IBM, June 2005.

[IET17a] IETF: Service Function Chaining (sfc). https://datatracker.ietf.org/wg/sfc/documents/, June 2017.

[IET17b] IRTF: Network Function Virtualization Research Group (NFVRG). https://irtf.org/nfvrg, June 2017.

[IFC00] Man page: ifconfig - configure a network interface. http://linuxcommand.org/man_pages/ifconfig8.html, August 2000.

[Ind16] M. Inden. *Der Java-Profi: Persistenzlösungen und REST-Services: Datenaustauschformate, Datenbankentwicklung und verteilte Anwendungen*. dpunkt.verlag, 2016.

[IP617] Man page: ip6tables - IPv6 packet filter administration. https://linux.die.net/man/8/ip6tables, March 2017.

[IPE17] iPerf - The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/, Jan 2017.

[IPF16] Man page: ipfw – User interface for firewall, traffic shaper, packet scheduler, in-kernel NAT. https://www.freebsd.org/cgi/man.cgi?query=ipfw, May 2016.

[ISO01] ISO. ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model. Technical report, International Organization for Standardization, 2001.

[ISO10] ISO/IEC. ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. Technical report, 2010.

[ISO11] ISO/IEC 20000-1:2011: Information technology – Service management – Part 1: Service management system requirements, Apr 2011.

[ITU98] ITU-T Recommendation G.841 - Types and characteristics of SDH network protection architectures, International Telecommunication Union. https://www.itu.int/rec/T-REC-G.841-199810-I/en, Oct 1998.

[ITU00] ITU-T Recommendation M.3400. https://www.itu.int/rec/T-REC-M.3400-200002-I/, Feb 2000.

[ITU07] ITU-T Rec. G.707/Y.1322, Network node interface for the synchronous digital hierarchy (SDH). https://www.itu.int/rec/T-REC-G.707/en, Jan 2007.

[ITU17] ITU Telecommunication Standardization Sector (ITU-T). www.itu.int, 2017.

[ITU92] Information Technology - Open Systems Interconnection - Systems Management: Alarm Reporting Function. https://www.itu.int/rec/T-REC-X.733-199202-I/en, Feb 92.

[JAD17] JAVA Agent DEvelopment Framework. http://jade.tilab.com/, 2017.

[Jam08] Mo Jamshidi. *Systems of Systems Engineering: Principles and Applications*. CRC Press, November 2008.

[JB00] Michael I. Jordan and Christopher M. Bishop. *An Introduction to Graphical Models*. 2000.

[Jen00] Nicholas R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 117(2):277–296, March 2000.

[JK14] Nachikethas A. Jagadeesan and Bhaskar Krishnamachari. Software-Defined Networking Paradigms in Wireless Networks: A Survey. *ACM Comput. Surv.*, 47(2):27:1–27:11, November 2014.

[JW96] N. Jennings and M. Wooldridge. Software agents. *IEE Review*, 42(1):17–20, Jan 1996.

[JWW94] Gabriel Jakobson, Robert Weihmayer, and Mark Weissman. *A Domain-Oriented Expert System Shell for Telecommunication Network Alarm Correlation*, pages 365–380. Springer US, Boston, MA, 1994.

[KAMH17] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke. A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements. *IEEE Access*, 5:1872–1899, Feb 2017.

[Kaw17] Christoph Kawan. Uniformly hyperbolic control theory. *Annual Reviews in Control*, 2017.

[KBA+99] David Kessens, Tony J. Bates, Cengiz Alaettinoglu, David Meyer, Curtis Villamizar, Marten Terpstra, Daniel Karrenberg, and Elise P. Gerich. Routing Policy Specification Language (RPSL). RFC 2622, June 1999.

[KBA+10] Vassilios Kaldanis, Péter Benkö, Domonkos Asztalos, Csaba Simon, Ranganai Chaparadza, and Giannis Katsaros. Methodology towards Integrating Scenarios and Testbeds for Demonstrating Autonomic/Self-managing Networks and Behaviors Required in Future Networks. In *Access Networks - 5th International ICST Conference on Access Networks, AccessNets 2010 and First ICST International Workshop on Autonomic Networking and Self-Management in Access Networks, SELFMAGICNETS 2010, Budapest, Hungary, November 3-5, 2010, Revised Selected Papers*, pages 240–252, 2010.

[KBMJ+08] Ethan Katz-Bassett, Harsha V. Madhyastha, John P. John, Arvind Krishnamurthy, David Wetherall, and Thomas Anderson. Studying Black Holes in the Internet with Hubble. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 247–262, Berkeley, CA, USA, 2008. USENIX Association.

[KC03] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[KD07] J. O. Kephart and R. Das. Achieving Self-Management via Utility Functions. *IEEE Internet Computing*, 11(1):40–48, Jan 2007.

[KF13] H. Kim and N. Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, February 2013.

[KKA17] Anand Jayant Kulkarni, Ganesh Krishnasamy, and Ajith Abraham. *Solution to 0–1 Knapsack Problem Using Cohort Intelligence Algorithm*, pages 55–74. Springer International Publishing, Cham, 2017.

[KKS02] Kee-Choon Kwon, Jin-Hyung Kim, and Poong-Hyun Seong. Hidden Markov model-based real-time transient identifications in nuclear power plants. *Int. J. Intell. Syst.*, 17(8):791–811, 2002.

[Koh90] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, Sep 1990.

[Kop82] Hermann Kopetz. The failure fault (ff) model. In *12th Symposium on Fault-Tolerant Computing*, pages 14–17, 1982.

[KRW12] Naga Praveen Katta, Jennifer Rexford, and David Walker. Logic Programming for Software-Defined Networks. http://frenetic-lang.org/publications/logic-programming-xldi12.pdf, 2012.

[KT16] M. A. Khan and H. Tembine. Autonomic management of future wireless networks. In *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 145–148, Dec 2016.

[KTP+10] Timotheos Kastrinogiannis, Nikolay Tcholtchev, Arun Prakash, Ranganai Chaparadza, Vassilios Kaldanis, Hakan Coskun, and Symeon Papavassiliou. Addressing Stability in Future Autonomic Networking. In *Mobile Networks and Management - Second International ICST Conference, MONAMI 2010, Santander, Spain, September 22-24, 2010, Revised Selected Papers*, pages 50–61, 2010.

[KvdHK+14] F. Kargl, R. W. van der Heijden, H. König, A. Valdes, and M. C. Dacier. Insights on the Security and Dependability of Industrial Control Systems. *IEEE Security Privacy*, 12(6):75–78, Nov 2014.

[KX02] M. K. Kona and Cheng-Zhong Xu. A framework for network management using mobile agents. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 8 pp–, April 2002.

[KYGS07] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. Detection and Localization of Network Black Holes. In *Proceedings of the IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pages 2180–2188, Washington, DC, USA, 2007. IEEE Computer Society.

[Lah00] Kevin Lahey. TCP Problems with Path MTU Discovery. RFC 2923, September 2000.

[LAK92] J.C. C. Laprie, A. Avizienis, and H. Kopetz, editors. *Dependability: Basic Concepts and Terminology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[Lap92] Jean-Claude Laprie. Dependability: A Unifying Concept for Reliable, Safe, Secure Computing. In *Algorithms, Software, Architecture - Information Processing '92, Volume 1, Proceedings of the IFIP 12th World Computer Congress, Madrid, Spain, 7-11 September 1992*, pages 585–593, 1992.

[LC15] Y. Li and M. Chen. Software-Defined Network Function Virtualization: A Survey. *IEEE Access*, 3:2542–2553, 2015.

[LCCG17] D.F. Link, C.G. Cordray, R.M. Chart, and K. Ginter. Management techniques for non-traditional network and information system topologies, January 2017. US Patent 9,537,731.

[LCP16] Z. Lu, G. Cao, and T. L. Porta. Networking smartphones for disaster recovery. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–9, March 2016.

[LCZ08] Nan Li, Guanling Chen, and Meiyuan Zhao. Autonomic fault management for wireless mesh networks - UMass Lowell. Technical report, 2008.

[LDA+05] Elyes Lehtihet, Hajer Derbel, Nazim Agoulmine, Yacine Ghamri-Doudane, and Sven van der Meer. Initial Approach Toward Self-configuration and Self-optimization in IP Networks. In *Proceedings of the 8th International Conference on Management of Multimedia Networks and Services*, MMNS'05, pages 371–382, Berlin, Heidelberg, 2005. Springer-Verlag.

[LFE+01] Steven D. Lind, Dino Farinacci, Richard Edmonstone, Jim Gemmell, Luigi Rizzo, and Michael Luby. PGM Reliable Transport Protocol Specification. RFC 3208, December 2001.

[LMM99] Meng Lin, Keith Marzullo, and Stefano Masini. Gossip Versus Deterministic Flooding: Low Message Overhead and High Reliability for Broadcasting on Small Networks. Technical report, La Jolla, CA, USA, 1999.

[LO15] Bob Lantz and Brian O'Connor. A Mininet-based Virtual Testbed for Distributed SDN Development. *SIGCOMM Comput. Commun. Rev.*, 45(4):365–366, August 2015.

[LRS+05] D. Larrabeiti, R. Romeral, I. Soto, M. Uruena, T. Cinkler, J. Szigeti, and J. Tapolcai. Multi-domain issues of resilience. In *Proc. International Conference on Transparent Optical Networks (ICTON)*, Barcelona, Spain, July 2005.

[LS98] C. Love and W. Siegel. *Understanding Token Ring Protocols and Standards*. Artech House, 1998.

[LSG16] J. De Lange, R. Von Solms, and M. Gerber. Information security management in local government. In *2016 IST-Africa Week Conference*, pages 1–11, May 2016.

[LWC+16] Jianjun Liu, Changzhi Wu, Jiang Cao, Xiangyu Wang, and Kok Lay Teo. A binary differential search algorithm for the 0–1 multidimensional knapsack problem. *Applied Mathematical Modelling*, 40(23–24):9788 – 9805, 2016.

[LWN+15] Eric Lui, Yu-Sung Wu, Patrick Ngai, Tung-Yueh Lin, Hong-Wei Li, and Shih-Yi Huang. Towards consistent software defined networking with logic programming. In *17th Asia-Pacific Network Operations and Management Symposium, APNOMS 2015, Busan, South Korea, August 19-21, 2015*, pages 109–114, 2015.

[LZGC08] B. Lang, N. Zhao, K. Ge, and K. Chen. An XACML Policy Generating Method Based on Policy View. In *2008 Third International Conference on Pervasive Computing and Applications*, volume 1, pages 295–301, Oct 2008.

[LZM+10] A. Liakopoulos, A. Zafeiropoulos, C. Marinos, M. Grammatikou, N. Tcholtchev, and P. Gouvas. Applying distributed monitoring techniques in autonomic networks. In *2010 IEEE Globecom Workshops*, pages 498–502, Dec 2010.

[MA17] Ahmed Mateen and Qaiser Abbas. IP Based Traffic Recovery: An Optimal Approach using SDN Application for Data Center Network. *CoRR*, abs/1702.00128, 2017.

[MAB+08] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[Mae17] Maestro Platform: A scalable control platform writen in Java which supports OpenFlow switches. https://zhengcai.github.io/maestro-platform/, Apr 2017.

[Man13] Eric Mannie. Generalized Multi-Protocol Label Switching (GMPLS) Architecture. RFC 3945, March 2013.

[Mar04] Miklós Maróti. *Directed Flood-Routing Framework for Wireless Sensor Networks*, pages 99–114. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[MAR+14] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. ClickOS and the Art of Network Function Virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 459–473, Berkeley, CA, USA, 2014. USENIX Association.

[MARH13] Joao Martins, Mohamed Ahmed, Costin Raiciu, and Felipe Huici. Enabling Fast, Dynamic Network Processing with clickOS. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 67–72, New York, NY, USA, 2013. ACM.

[MAT16] MATLAB. https://www.mathworks.com/products/matlab.html, Dec 2016.

[MBKM17] D. L. C. Mack, G. Biswas, X. D. Koutsoukos, and D. Mylaraswamy. Learning Bayesian Network Structures to Augment Aircraft Diagnostic Reference Models. *IEEE Transactions on Automation Science and Engineering*, 14(1):358–369, Jan 2017.

[MCR+16] Tayeb Ben Meriem, Ranganai Chaparadza, Benoît Radier, Said Soulhi, Jose-Antonio Lozano-Lopez, and Arun Prakash. *ETSI White Paper No. 16, GANA - Generic Autonomic Networking Architecture; Reference Model for Autonomic Networking, Cognitive Networking and Self-Management of Networks and Services*. ETSI, European Telecommunications Standards Institute, 1nd edition, October 2016.

[MD90] J.C. Mogul and S.E. Deering. Path MTU discovery. RFC 1191 (Draft Standard), November 1990.

[MESW01] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model – Version 1 Specification. RFC 3060 (Proposed Standard), February 2001. Updated by RFC 3460.

[MGSF16] C. C. Machado, L. Z. Granville, and A. Schaeffer-Filho. Answer: Combining nfv and sdn features for network resilience strategies. In *2016 IEEE Symposium on Computers and Communication (ISCC)*, pages 391–396, June 2016.

[MHFv16] Jedidiah McClurg, Hossein Hojjat, Nate Foster, and Pavol Černý. Event-driven Network Programming. *SIGPLAN Not.*, 51(6):369–385, June 2016.

[MIB+08] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. N. Chuah, Y. Ganjali, and C. Diot. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Transactions on Networking*, 16(4):749–762, Aug 2008.

[MII00] Man page: mii-tool - view, manipulate media-independent interface status. https://www.netadmintools.com/html/mii-tool.man.html, April 2000.

[Min17] Mininet: An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/, Apr 2017.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '01, pages 346–, Washington, DC, USA, 2001. IEEE Computer Society.

[MMMS16] F. Morone, L. Ma, H. Makse, and A. Scala. Enhancing network resilience via self-healing. In *2016 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*, pages 1–5, June 2016.

[MMS+17] P. Maciel, R. Matos, B. Silva, J. Figueiredo, D. Oliveira, I. Fé, R. Maciel, and J. Dantas. Mercury: Performance and Dependability Evaluation of Systems with Exponential, Expolynomial, and General Distributions. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 50–57, Jan 2017.

[Mor11] K. T. Morrison. Rapidly recovering from the catastrophic loss of a major telecommunications office. *IEEE Communications Magazine*, 49(1):28–35, January 2011.

[Moy98] John T. Moy. OSPF Version 2. RFC 2328, April 1998.

[MRF+13] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing Software-defined Networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[MSBY16] Quang Tran Minh, Yoshitaka Shibata, Cristian Borcea, and Shigeki Yamada. On-site configuration of disaster recovery access networks made easy. *Ad Hoc Networks*, 40:46 – 60, 2016.

[MSG+16] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, Firstquarter 2016.

[MSW16] M. McCormick, C.S. Saunderson, and R. Whinnery. System and method for improving recovery of a telecommunications network from an unscheduled loss of service using repeatable requirements for applications by design criticality classification, Nov 2016. US Patent 9,483,743.

[Mus15] Mark A. Musen. The ProtÉGÉ Project: A Look Back and a Look Forward. *AI Matters*, 1(4):4–12, June 2015.

[MYS17] MySQL. https://www.mysql.com/, Jan 2017.

[Nal05] M. Naldi. Connectivity of Waxman Topology Models. *Comput. Commun.*, 29(1):24–31, December 2005.

[Nat12] Mary Natrella. *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST/SEMATECH, 2012.

[NBM15] T.P. Khanh Nguyen, Julie Beugin, and Juliette Marais. Method for evaluating an extended fault tree to analyse the dependability of complex systems: Application to a satellite-based railway system. *Reliability Engineering & System Safety*, 133:300 – 313, 2015.

[NCC+16] Pedro Neves, Rui Calé, Mário Rui Costa, Carlos Parada, Bruno Parreira, Jose Alcaraz-Calero, Qi Wang, James Nightingale, Enrique Chirivella-Perez, Wei Jiang, Hans Dieter Schotten, Konstantinos Koutsopoulos, Anastasius Gavras, and Maria João Barros. The SELFNET Approach for Autonomic Management in an NFV/SDN Networking Paradigm. *Int. J. Distrib. Sen. Netw.*, 2016:2:2–2:2, January 2016.

[NCM16] Roberto Natella, Domenico Cotroneo, and Henrique S. Madeira. Assessing dependability with software fault injection: A survey. *ACM Comput. Surv.*, 48(3):44:1–44:55, February 2016.

[Nea03] Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[NET17] The netfilter.org project. https://www.netfilter.org/, 2017.

[NFV12] Network Functions Virtualisation – Introductory White Paper. https://portal.etsi.org/NFV/NFV_White_Paper.pdf, Oct 2012.

[NHG+16] Gianfranco Nencioni, Bjarne E. Helvik, Andrés J. Gonzalez, Poul E. Heegaard, and Andrzej Kamisinski. Availability Modelling of Software-Defined Backbone Networks. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2016, Toulouse, France, June 28 - July 1, 2016*, pages 105–112, 2016.

[NIC15] NICE. https://github.com/mcanini/nice/wiki, Aug 2015.

[NLO16] NLOPT library. http://ab-initio.mit.edu/wiki/index.php/NLopt, Dec 2016.

[NOX17] The NOX Controller. https://github.com/noxrepo/nox, Apr 2017.

[NRL09] Mikko Nieminen, Tomi Räty, and Mikko Lindholm. Multi-sensor Logical Decision Making in the Single Location Surveillance Point System. In *The Fourth International Conference on Systems, ICONS 2009, Gosier, Guadeloupe, France, 1-6 March 2009*, pages 86–90, 2009.

[NRS+11] Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Corrêa, Sidney C. de Lucena, and Maurício F. Magalhães. Virtual Routers As a Service: The RouteFlow Approach Leveraging Software-defined Networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, CFI '11, pages 34–37, New York, NY, USA, 2011. ACM.

[NS07] Nicholas Nethercote and Julian Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. *SIGPLAN Not.*, 42(6):89–100, June 2007.

[NTSR14] Mikko Nieminen, Nikolay Tcholtchev, Ina Schieferdecker, and Tomi Räty. Robust architecture for distributed intelligence in an IP-based mobile wide-area surveillance system. *The Journal of Supercomputing*, 70(3):1120–1141, 2014.

[OAS17] OASIS eXtensible Access Control Markup Language (XACML) TC. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2017.

[OCT16] GNU Octave. http://wiki.octave.org, Dec 2016.

[OMG04] OMG IDL Syntax and Semantics. http://www.omg.org/cgi-bin/doc?formal/02-06-39.pdf, Jul 2004.

[ONF12] Software-Defined Networking: The New Norm for Networks. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf, April 2012.

[ONF17] ONF: Open Networking Foundation. https://www.opennetworking.org/, Apr 2017.

[Ope11] OpenFlow Reference Implementation. http://www.openflow.org/wp/downloads/, 2011.

[Ope16] Open Virtual Switch. http://vswitch.org, 2016.

[Ope17a] OpenAz. http://incubator.apache.org/projects/openaz.html, Apr 2017.

[Ope17b] OpenFlow. https://www.opennetworking.org/sdn-resources/openflow, Apr 2017.

[OPE17c] OpenNMS. https://www.opennms.org, Jan 2017.

[OWM11] J. C. Oberg, A. G. Whitt, and R. M. Mills. Disasters will happen - are you ready? *IEEE Communications Magazine*, 49(1):36–42, January 2011.

[Pap03] Christos H. Papadimitriou. Computational Complexity. In *Encyclopedia of Computer Science*, pages 260–265. John Wiley and Sons Ltd., Chichester, UK, 2003.

[Pas16] Adrian Paschke. Provalets: Component-based mobile agents as microservices for rule-based data access, processing and analytics. *Business & Information Systems Engineering*, 58(5):329–340, Oct 2016.

[PBR17] Software Configuration Guide—Release 12.2, Catalyst 4500, Chapter 25: Configuring Policy-Based Routing. http://www.cisco.com/c/en/us/support/docs/security/ios-firewall/23602-confaccesslists.pdf, 2017.

[PCA17] Man page: PCAP - Packet Capture library. http://www.tcpdump.org/manpages/pcap.3pcap.html, January 2017.

[PCS11] A. Prakash, R. Chaparadza, and A. Starschenko. A Model-driven approach to design and verify autonomic network behaviors. In *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, pages 701–706, Dec 2011.

[PER17] perfSONAR. http://www.perfsonar.net/, Jan 2017.

[Pil04] A. Pilz. "Policy-Maker": a toolkit for policy-based security management. In *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No.04CH37507)*, volume 1, pages 263–276 Vol.1, April 2004.

[PIN17] Man page: ping, ping6 - send ICMP ECHO_REQUEST to network hosts. https://linux.die.net/man/8/ping, Jan 2017.

[PLG+16] M. Pourvali, K. Liang, F. Gu, H. Bai, K. Shaban, S. Khan, and N. Ghani. Progressive recovery for network virtualization after large-scale disasters. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pages 1–5, Feb 2016.

[PMT16] C. V. Phung, Q. T. Minh, and M. Toulouse. Routing Optimization Model in Multihop Wireless Access Networks for Disaster Recovery. In *2016 International Conference on Advanced Computing and Applications (ACOMP)*, pages 135–140, Nov 2016.

[PNL03] Probabilistic Network Library: User Guide and Reference Manual. 2003.

[PNL16] Probabilistic Network Library: PNL. http://sourceforge.net/projects/openpnl/, Dec 2016.

[Pon17] Ponder2. http://ponder2.net/, Apr 2017.

[Pos81] J. Postel. Internet Control Message Protocol. RFC 792, September 1981.

[Pos17] PostgreSQL. https://www.postgresql.org/, Jan 2017.

[Pow98] M. J. D. Powell. Direct Search Algorithms for Optimization Calculations. *Acta Numerica*, 7:287–336, 1998.

[POX17] The POX Controller. https://github.com/noxrepo/pox, Apr 2017.

[PRO13] Prova Rule Language. https://prova.ws/, Jan 2013.

[PRO17] Protégé: A free, open-source ontology editor and framework for building intelligent systems. http://protege.stanford.edu/, Jan 2017.

[PRS16] Vitor Pereira, Miguel Rocha, and Pedro Sousa. *Automated Network Resilience Optimization Using Computational Intelligence Methods*, pages 485–495. Springer International Publishing, Cham, 2016.

[PTC12] Arun Prakash, Zoltán Theisz, and Ranganai Chaparadza. *Formal Methods for Modeling, Refining and Verifying Autonomic Components of Computer Networks*, pages 1–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[PvHG14] T. Pitakrat, A. v. Hoorn, and L. Grunske. Increasing dependability of component-based software systems by online failure prediction (short paper). In *2014 Tenth European Dependable Computing Conference*, pages 66–69, May 2014.

[QMSW17] J. Qin, Q. Ma, Y. Shi, and L. Wang. Recent Advances in Consensus of Multi-Agent Systems: A Brief Survey. *IEEE Transactions on Industrial Electronics*, 64(6):4972–4983, June 2017.

[QUA17] Quagga Routing Suite. http://www.nongnu.org/quagga/, Jan 2017.

[Rab90] Lawrence R. Rabiner. Readings in Speech Recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[Ran98] B. Randell. Dependability-a unifying concept. In *Proceedings Computer Security, Dependability, and Assurance: From Needs to Solutions (Cat. No.98EX358)*, pages 16–25, 1998.

[Ran00] Brian Randell. Facing up to faults. *The Computer Journal*, 43:95–106, 2000.

[Ran11] Y. Ran. Considerations and suggestions on improvement of communication network disaster countermeasures after the wenchuan earthquake. *IEEE Communications Magazine*, 49(1):44–47, January 2011.

[RASM17] Saeid Rastegar, Rui Araújo, Jalil Sadati, and Jérôme Mendes. A novel robust control scheme for {LTV} systems using output integral discrete-time synergetic control theory. *European Journal of Control*, 34:39 – 48, 2017.

[RBA10] XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile Version 1.0. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-rbac-v1-spec-cd-03-en.html, March 2010.

[Res17] ResiliNets project definitions. https://wiki.ittc.ku.edu/resilinets_wiki/index.php/Definitions, April 2017.

[RFGS10] Carlos Rodriguez-Fernández and Jorge Jesús Gómez-Sanz. Self-management Capability Requirements with SelfMML & INGENIAS to Attain Self-organising Behaviours. In *Proceedings of the Second International Workshop on Self-organizing Architectures*, SOAR '10, pages 11–20, New York, NY, USA, 2010. ACM.

[RFR+12] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for Network Update. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 323–334, New York, NY, USA, 2012. ACM.

[RFRW11] Mark Reitblatt, Nate Foster, Jennifer Rexford, and David Walker. Consistent Updates for Software-defined Networks: Change You Can Believe in! In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 7:1–7:6, New York, NY, USA, 2011. ACM.

[RHLP16] Alban Rousset, Bénédicte Herrmann, Christophe Lang, and Laurent Philippe. A survey on parallel and distributed multi-agent systems for high performance computing simulations. *Computer Science Review*, 22:27 – 46, 2016.

[Rij79] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[RMD05] S. Rai, B. Mukherjee, and O. Deshpande. IP resilience within an autonomous system: current approaches, challenges, and future directions. *IEEE Communications Magazine*, 43(10):142–149, Oct 2005.

[RNCS09] Gábor Rétvári, Felician Németh, Ranganai Chaparadza, and Róbert Szabó. OSPF for Implementing Self-adaptive Routing in Autonomic Networks: A Case Study. In *Modelling Autonomic Communications Environments, Fourth IEEE International Workshop, MACE 2009, Venice, Italy, October 26-27, 2009. Proceedings*, pages 72–85, 2009.

[Rou17] RouteFlow. http://routeflow.github.io/RouteFlow/, Apr 2017.

[RPNR17] Jacek Rak, Dimitri Papadimitriou, Heiko Niedermayer, and Pablo Romero. Information-driven network resilience: Research challenges and perspectives. *Optical Switching and Networking*, 23, Part 2:156 – 178, 2017. Design and modeling of Resilient optical networks {RNDM} 2015.

[RPR16] C. Ramirez-Perez and V. Ramos. SDN meets SDR in self-organizing networks: fitting the pieces of network management. *IEEE Communications Magazine*, 54(1):48–57, January 2016.

[RTCM11] Y. Rebahi, N. Tcholtchev, R. Chaparadza, and V. N. Merekoulias. Addressing security issues in the autonomic Future Internet. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 517–518, Jan 2011.

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[RY05] Thomas Philip Runarsson and Xin Yao. Search biases in constrained evolutionary optimization. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 35(2):233–243, 2005.

[Sah07] Goutam Kumar Saha. Self-healing Software. *Ubiquity*, 2007(March):4:1–4:1, March 2007.

[SAL06] John Strassner, N. Agoulmine, and E. Lehtihet. FOCALE: A Novel Autonomic Networking Architecture. 2006.

[Sam16] Sami J. Habib and Paulvanna Nayaki Marimuthu. Managing Enterprise Network Resilience Through the Mimicking of Bio-Organisms. In *New Advances in Information Systems and Technologies - Volume 1 [WorldCIST'16, Recife, Pernambuco, Brazil, March 22-24, 2016].*, pages 901–910, 2016.

[SB13] Florian Schneider and Brian Berenbach. A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, 16:796 – 805, 2013.

[SBD+17] Thiago Santini, Christoph Borchert, Christian Dietrich, Horst Schirmeier, Martin Hoffmann, Olaf Spinczyk, Daniel Lohmann, Flávio Rech Wagner, and Paolo Rech. *Effectiveness of Software-Based Hardening for Radiation-Induced Soft Errors in Real-Time Operating Systems*, pages 3–15. Springer International Publishing, Cham, 2017.

[SBPM09] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[SC05] Stanislav Shalunov and Richard Carlson. *Detecting Duplex Mismatch on Ethernet*, pages 135–148. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[SC15] Sukhpal Singh and Inderveer Chana. Qos-aware autonomic resource management in cloud computing: A systematic review. *ACM Comput. Surv.*, 48(3):42:1–42:46, December 2015.

[Sch11] Boris Schling. *The Boost C++ Libraries*. XML Press, 2011.

[Sch12] Rainer Schlittgen. *Einführung in die Statistik. Analyse und Modellierung von Daten*. Oldenbourg Verlag, München, Germany, 2012.

[SCU16] S-Cube, the European Network of Excellence in Software Services and Systems. http://www.s-cube-network.eu, dec 2016.

[SFR03] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming, Vol. 1*. Pearson Education, 3 edition, 2003.

[SHÇ+10] James P. G. Sterbenz, David Hutchison, Egemen K. Çetinkaya, Abdul Jabbar, Justin P. Rohrer, Marcus Schöller, and Paul Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245–1265, 2010.

[She07] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 edition, 2007.

[She09] Haibo Shen. A Semantic-Aware Attribute-Based Access Control Model for Web Services. In Arrems Hua and Shih-Liang Chang, editors, *ICA3PP*, volume 5574 of *Lecture Notes in Computer Science*, pages 693–703. Springer, 2009.

[Shi07] Robert W. Shirey. Internet Security Glossary, Version 2. RFC 4949, August 2007.

[SID17a] GB922 Information Framework (SID) R16.5.0. https://www.tmforum.org/resources/suite/gb922-information-framework-sid-r16-5-0/, Jan 2017.

[SID17b] The Information Framework (SID). https://www.tmforum.org/information-framework-sid, Jan 2017.

[SJ17] Sanjeev Singh and Rakesh Kumar Jha. A Survey on Software Defined Networking: Architecture for Next Generation Network. *J. Netw. Syst. Manage.*, 25(2):321–374, April 2017.

[SKGR15] S. Sun, M. Kadoch, L. Gong, and B. Rong. Integrating network function virtualization with SDR and SDN for 4G/5G networks. *IEEE Network*, 29(3):54–59, May 2015.

[SLBK13] Richard William Skowyra, Andrei Lapets, Azer Bestavros, and Assaf Kfoury. Verifiably-safe Software-defined Networks for CPS. In *Proceedings of the 2Nd ACM International Conference on High Confidence Networked Systems*, HiCoNS '13, pages 101–110, New York, NY, USA, 2013. ACM.

[SM03] Ravi Sahita and Keith McCloghrie. Framework Policy Information Base. RFC 3318, March 2003.

[SMA+16] K. Safi, S. Mohammed, F. Attal, M. Khalil, and Y. Amirat. Recognition of different daily living activities using hidden Markov model regression. In *2016 3rd Middle East Conference on Biomedical Engineering (MECBME)*, pages 16–19, Oct 2016.

[SMTZ13] B. Silva, P. Maciel, E. Tavares, and A. Zimmermann. Dependability models for designing disaster tolerant cloud computing systems. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–6, June 2013.

[SPS00] Gerald L. Thompson Suresh P. Sethi. *Optimal Control Theory: Applications to Management Science and Economics*. Springer US, 2000.

[SRKO16] Mehdi Mohammad Salehi, Mehdi Rahmati, Masoud Karimnezhad, and Pouria Omidvar. Estimation of the non records logs from existing logs using artificial neural networks. *Egyptian Journal of Petroleum*, pages –, 2016.

[SS04a] Malgorzata Steinder and Adarshpal S. Sethi. A survey of fault localization techniques in computer networks. *Sci. Comput. Program.*, 53(2):165–194, 2004.

[SS04b] Malgorzata Steinder and Adarshpal S. Sethi. Probabilistic Fault Localization in Communication Systems Using Belief Networks. *IEEE/ACM Trans. Netw.*, 12(5):809–822, October 2004.

[SS04c] G. Suwala and G. Swallow. SONET/SDH-like resilience for IP networks: a survey of traffic protection mechanisms. *IEEE Network*, 18(2):20–25, Mar 2004.

[Sta98] William Stallings. *SNMP,SNMPV2,Snmpv3,and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1998.

[STP+15] Alexej Starschenko, Nikolay Tcholtchev, Arun Prakash, Ina Schieferdecker, and Ranganai Chaparadza. Auto-configuration of OSPFv3 routing in fixed IPv6 networks. In *7th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops, ICUMT 2015, Brno, Czech Republic, October 6-8, 2015*, pages 196–205, 2015.

[Str02] J. Strassner. DEN-ng: achieving business-driven network management. In *NOMS 2002. IEEE/IFIP Network Operations and Management Symposium. ' Management Solutions for the New Communications World'(Cat. No.02CH37327)*, pages 753–766. IEEE, August 2002.

[Str03a] John Strassner. *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[Str03b] John Strassner. *Policy-Based Network Management: Solutions for the Next Generation (The Morgan Kaufmann Series in Networking)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[SWC+] Szymon Szott, Michal Wódczak, Ranganai Chaparadza, Tayeb Ben Meriem, Kostas Tsagkaris, Apostolos Kousaridas, Benoit Radier, Andrej Mihailovic, Marek Natkaniec, Krzysztof Loziak, Katarzyna Kosek-Szott, and Michal Wagrowski. Standardization of an autonomicity-enabled mesh architecture framework, from ETSI-AFI group perspective: Work in progress (Part 2 of 2). In *Workshops Proceedings of the Global Communications Conference, GLOBECOM 2012, 3-7 December 2012, Anaheim, California, USA*.

[SWC+12] Szymon Szott, Michal Wódczak, Ranganai Chaparadza, Tayeb Ben Meriem, Kostas Tsagkaris, Apostolos Kousaridas, Benoit Radier, Andrej Mihailovic, Marek Natkaniec, Krzysztof Loziak, Katarzyna Kosek-Szott, and Michal Wagrowski. Standardization of an autonomicity-enabled mesh architecture framework, from ETSI-AFI group perspective: Work in progress (Part 1 of 2). In *Workshops Proceedings of the Global Communications Conference, GLOBECOM 2012, 3-7 December 2012, Anaheim, California, USA*, pages 847–851, 2012.

[SWLVH13] D. Schütz, A. Wannagat, C. Legat, and B. Vogel-Heuser. Development of PLC-Based Software for Increasing the Dependability of Production Automation Systems. *IEEE Transactions on Industrial Informatics*, 9(4):2397–2406, Nov 2013.

[SYS17] Man page: sysklogd - Linux system logging utilities. https://linux.die.net/man/8/syslogd, 2017.

[TC08] Editors: Nikolay Tcholtchev and Ranganai Chaparadza. FP6-IST-27489/WP3/D.3.9v1: Implementation of Selected Components of the Failure Detection and Fault Management part of the ANA architecture . Dec 2008.

[TC10a] N. Tcholtchev and R. Chaparadza. Autonomic Fault-Management and resilience from the perspective of the network operation personnel. In *2010 IEEE Globecom Workshops*, pages 469–474, Dec 2010.

[TC10b] Nikolay Tcholtchev and Ranganai Chaparadza. On Self-healing Based on Collaborating End-Systems, Access, Edge and Core Network Components. In *Access Networks - 5th International ICST Conference on Access Networks, AccessNets 2010 and First ICST International Workshop on Autonomic Networking and Self-Management in Access Networks, SELFMAGICNETS 2010, Budapest, Hungary, November 3-5, 2010, Revised Selected Papers*, pages 283–298, 2010.

[TCC10] Nikolay Tcholtchev, Agnieszka Betkowska Cavalcante, and Ranganai Chaparadza. Scalable Markov Chain Based Algorithm for Fault-Isolation in Autonomic Networks. In *Proceedings of the Global Communications Conference, 2010. GLOBECOM 2010, 6-10 December 2010, Miami, Florida, USA*, pages 1–6, 2010.

[TÇE17] LanAnh Trinh, Baran Çürüklü, and Mikael Ekström. Fault Tolerance Analysis for Dependable Autonomous Agents Using Colored Time Petri Nets. In *9th International Conference on Agents and Artificial Intelligence 2017*. Springer, January 2017.

[TCG17] TCG: Trusted Computing Group. http://www.trustedcomputinggroup.org/, 2017.

[Tch09] N. Tcholtchev. Components and Mechanisms of Autonomic Fault-Management for Self-Managing Networks. *Imprint Berlin, TU Berlin, Dipl.-Arb.*, pages 1–139, January 2009.

[Tch10] Editor: Nikolay Tcholtchev. INFSO-ICT-215549-EFIPSANS-WP4-D4.5: Components and Mechanisms for Autonomic Fault-Management. Jan 2010.

[TCP09] Nikolay Tcholtchev, Ranganai Chaparadza, and Arun Prakash. Addressing Stability of Control-Loops in the Context of the GANA Architecture: Synchronization of Actions and Policies. In *Self-Organizing Systems, 4th IFIP TC 6 International Workshop, IWSOS 2009, Zurich, Switzerland, December 9-11, 2009. Proceedings*, pages 262–268, 2009.

[TCP15] Man page: tcpdump - dump traffic on a network . http://www.tcpdump.org/tcpdump_man.html, Sept 2015.

[TCP16] Tcptrace. http://www.tcptrace.org/, Jan 2016.

[TCRMDlF16] Esteban Tlelo-Cuautle, José de Jesús Rangel-Magdaleno, and Luis Gerardo De la Fraga. *Matlab-Simulink Co-Simulation*, pages 61–75. Springer International Publishing, Cham, 2016.

[TDM16] Huy T. Tran, Jean Charles Domerçant, and Dimitri N. Mavris. A Network-based Cost Comparison of Resilient and Robust System-of-Systems. *Procedia Computer Science*, 95:126 – 133, 2016.

[TDW+14] N. Tcholtchev, G. Dudeck, M. Wagner, C. Hein, A. Prakash, and T. Ritter. Integrating the Modelica DSL into a Platform for Model-Based Tool Interoperability. In *2014 IEEE 38th International Computer Software and Applications Conference Workshops*, pages 528–534, July 2014.

[TEL09] Telcordia GR-253-CORE, Synchronous Optical Network (SONET) Transport Systems: Common Generic Criteria, Issue 5, 2009.

[Teo09] Iuliana Teodorescu. Maximum Likelihood Estimation for Markov Chains. May 2009.

[TGSB15] Matthias C. M. Troffaes, Jacob Gledhill, Damjan Skulj, and Simon Blake. Using imprecise continuous time Markov chains for assessing the reliability of power networks with common cause failure and non-immediate repair. In Thomas Augustin, Serena Doria, Enrique Miranda, and Erik Quaeghebeur, editors, *ISIPTA '15: Proceedings of the Ninth International Symposium on Imprecise Probability: Theories and Applications*, pages 287–294, 2015.

[TGV09] Nikolay Tcholtchev, Monika Grajzer, and Bruno Vidalenc. Towards a Unified Architecture for Resilience, Survivability and Autonomic Fault-Management for Self-managing Networks. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops - International Workshops, ICSOC/ServiceWave 2009, Stockholm, Sweden, November 23-27, 2009, Revised Selected Papers*, pages 335–344, 2009.

[TH01] F. Touvet and D. Harle. *Network Resilience in Multilayer Networks: A Critical Review and Open Issues*, pages 829–837. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[Tiv16] Tivoli Intelligent Orchestrator. https://www-01.ibm.com/marketing/iwm/tnd/demo.jsp?id=Tivoli+Intelligent+Orchestrator+Jan05, Dec 2016.

[TK04] Gerald Tesauro and Jeffrey O. Kephart. Utility Functions in Autonomic Systems. In *Proceedings of the First International Conference on Autonomic Computing*, ICAC '04, pages 70–77, Washington, DC, USA, 2004. IEEE Computer Society.

[TMF17] TMF: TeleManagement Forum. http://www.tmforum.org/, 2017.

[TOC06] Stuart R. Thorncraft, Hugh R. Outhred, and David J. Clements. Evaluation of Open-Source LP Optimization Codes in Solving Electricity Spot Market Optimization Problems. In *19th Mini-Euro Conference on Operation Research Models and Methods in the Energy Sector, Coimbra, Portugal*, Sep 2006.

[TOP17] Man page: top - display Linux tasks. https://linux.die.net/man/1/top, Jan 2017.

[TOU17] Man page: touch - change file timestamps. http://man7.org/linux/man-pages/man1/touch.1.html, Jan 2017.

[TP12] Nikolay Tcholtchev and Razvan Petre. Design, Implementation and Evaluation of a Framework for Adaptive Monitoring in IP Networks. pages 1–10, 2012.

[TPS+12] Nikolay Tcholtchev, Arun Prakash, Ina Schieferdecker, Ranganai Chaparadza, and Razvan Petre. Auto-Collaboration for optimal network resource utilization in fixed IPv6 networks. In *Workshops Proceedings of the Global Communications Conference, GLOBECOM 2012, 3-7 December 2012, Anaheim, California, USA*, pages 807–812, 2012.

[TRA17] Man page: traceroute - print the route packets trace to network host. https://linux.die.net/man/8/traceroute, Jan 2017.

[Tre17] Full-Stack OpenFlow Framework in Ruby and C. http://trema.github.io/trema/, Apr 2017.

[TS14a] Nikolay Tcholtchev and Ina Schieferdecker. Framework for distributed autonomic self-healing in fixed IPv6 networks. *Int. J. Communication Systems*, 27(12):4103–4125, 2014.

[TS14b] Nikolay Tcholtchev and Ina Schieferdecker. Framework for Ensuring Runtime Stability of Control Loops in Multi-agent Networked Environments. *Trans. Computational Science*, 22:64–92, 2014.

[TZ04] Paul Weber Tammy Zitello, Deborah Williams. *HP OpenView System Administration Handbook: Network Node Manager, Customer Views, Service Information Portal, OpenView Operations*. Prentice Hall, 2004.

[Uni17] UniverSelf. http://www.univerself-project.eu/, 2017.

[VAL16] Valgrind. http://valgrind.org/, Dec 2016.

[Vas09] Emil Vassev. *ASSL - Autonomic System Specification Language: A Framework for Specification and Code Generation of Autonomic Systems*. LAP Lambert Academic Publishing, Germany, 2009.

[VCF+00] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA Authorization Framework. RFC 2904 (Informational), August 2000.

[VCNR13] Bruno Vidalenc, Laurent Ciavaglia, Ludovic Noirie, and Eric Renault. Dynamic risk-aware routing for OSPF networks. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, May 27-31, 2013*, pages 226–234, 2013.

[VH08] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[VH12] Emil Vassev and Mike Hinchey. The ASSL Approach to Specifying Self-managing Embedded Systems. *Concurr. Comput. : Pract. Exper.*, 24(16):1860–1878, November 2012.

[VNC12] B. Vidalenc, L. Noirie, and L. Ciavaglia. GMPLS adaptive level of recovery. In *2012 IEEE International Conference on Communications (ICC)*, pages 2735–2740, June 2012.

[VPD04] Jean-Philippe Vasseur, Mario Pickavet, and Piet Demeester. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[vW17] Wessel N. van Wieringen. On the mean squared error of the ridge estimator of the covariance and precision matrix. *Statistics & Probability Letters*, 123:88 – 92, 2017.

[Wal09] Stefan Wallin. Chasing a Definition of "Alarm". *J. Netw. Syst. Manage.*, 17(4):457–481, December 2009.

[WCLL08] Fang Wang, Shanzhi Chen, Xin Li, and Yuhong Li. A Route Flap Suppression Mechanism Based on Dynamic Timers in OSPF Network. In *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008, Zhang Jia Jie, Hunan, China, November 18-21, 2008*, pages 2154–2159, 2008.

[WCW+98] Zheng Wang, Mark A. Carlson, Walter Weiss, Elwyn B. Davies, and Steven L. Blake. An Architecture for Differentiated Services. RFC 2475, December 1998.

[Wer07] Matthias Werner. *Eigenschaften verlässlicher Systeme*. habilitation, Berlin University of Technology, 2007.

[Whi08] Juniper Network Whitepaper. What's Behind Network Downtime? Proactive Steps to Reduce Human Error and Improve Availability of Networks. Technical report, Juniper Networks Inc., 2008.

[WL09] Stefan Wallin and Viktor Leijon. *Telecom Network and Service Management: An Operator Survey*, pages 15–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[WMR+11] Michal Wodczak, Tayeb Ben Meriem, Benoit Radier, Ranganai Chaparadza, Kevin Quinn, Jesse Kielthy, Brian A. Lee, Laurent Ciavaglia, Kostas Tsagkaris, Szymon Szott, Anastasios Zafeiropoulos, Athanassios Liakopoulos, Apostolos Kousaridas, and Maurice Duault. Standardizing a Reference Model and Autonomic Network Architectures for the self-managing Future Internet. *IEEE Network*, 25(6):50–56, 2011.

[WRH+15] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang. Toward a software-based network: integrating software defined networking and network function virtualization. *IEEE Network*, 29(3):36–41, May 2015.

[Wro97] John T. Wroclawski. The Use of RSVP with IETF Integrated Services. RFC 2210, September 1997.

[WS04] A. Westerinen and J. Schott. Implementation of the CIM Policy Model using PONDER. In *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.*, pages 207–210, June 2004.

[WSS+01] Andrea Westerinen, John Schnizlein, John Strassner, Mark Scherling, Bob Quinn, Jay Perry, Shai Herzog, An-Ni Huynh, Mark Carlson, and Steve Waldbusser. Terminology for Policy-Based Management. Internet RFC 3198, November 2001.

[WTVL13] Michal Wódczak, Nikolay Tcholtchev, Bruno Vidalenc, and Yuhong Li. Design and Evaluation of Techniques for Resilience and Survivability of the Routing Node. *IJARAS*, 4(4):36–63, 2013.

[WZC08] Yi Wang, Nevin L. Zhang, and Tao Chen. Latent Tree Models and Approximate Inference in Bayesian Networks. *J. Artif. Int. Res.*, 32(1):879–900, August 2008.

[XAC17] OASIS eXtensible Access Control Markup Language (XACML). https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml, 2017.

[XAY+16] L. Xu, H. Assem, I. G. B. Yahia, T. S. Buda, A. Martin, D. Gallico, M. Biancani, A. Pastor, P. A. Aranda, M. Smirnov, D. Raz, O. Uryupina, A. Mozo, B. Ordozgoiti, M. I. Corici, P. O'Sullivan, and R. Mullins. CogNet: A network management architecture featuring cognitive capabilities. In *2016 European Conference on Networks and Communications (EuCNC)*, pages 325–329, June 2016.

[XEn12] XEngine Project. http://xacmlpdp.sourceforge.net/, 2008-2012.

[XER17] The Apache Xerces Project. http://xerces.apache.org/, 2017.

[XG04] Yangsheng Xu and Ming Ge. Hidden markov model-based process monitoring system. *Journal of Intelligent Manufacturing*, 15(3):337–350, 2004.

[XHLH13] X. Xu, Z. Hou, C. Lian, and H. He. Online Learning Control Using Adaptive Critic Designs With Sparse Kernel Machines. *IEEE Transactions on Neural Networks and Learning Systems*, 24(5):762–775, May 2013.

[XOR17] XORP: eXtensible Open Router Platform. http://www.xorp.org/, Jan 2017.

[XPA17] XPath specification. http://www.w3.org/TR/xpath, Jan 2017.

[YFN+14] Vitaliy Yakovyna, Dmytro Fedasyuk, Oksana Nytrebych, Iurii Parfenyuk, and Volodymyr Matselyukh. Software Reliability Assessment Using High-Order Markov Chains. *International Journal of Engineering Science Invention*, 3(7), July 2014.

[YJH+16] S. Yin, Chen Jing, Jian Hou, O. Kaynak, and H. Gao. PCA and KPCA integrated Support Vector Machine for multi-fault classification. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 7215–7220, Oct 2016.

[YLTP16] Lingjian Yang, Songsong Liu, Sophia Tsoka, and Lazaros G. Papageorgiou. Mathematical programming for piecewise linear regression analysis. *Expert Systems with Applications*, 44:156 – 167, 2016.

[YS09] Xin Yan and Xiao Gang Su. *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2009.

Appendix A

ISO/OSI, TCP/IP and Common Network Technologies

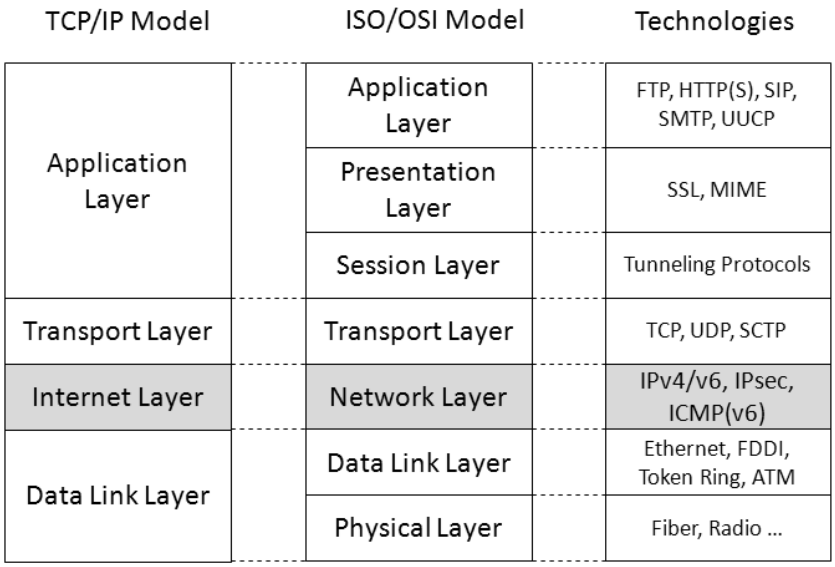

Figure A.1: Illustrative Comparison of the ISO/OSI and TCP/IP Reference Models

Appendix B

Additional FDH Dynamic Aspects

The following section focuses on the initialization and bootstrapping of the FDH components within a node. This is a very vital aspect of the proposed framework. However, since it is not at the heart of the designed algorithmic frameworks and control loops, it was not presented in the main design and specification part of the thesis (section 6.7).

B.1 Initialization of the FDH Components on Node Level

The overall initialization of the FDH components on node level is described in a set of sequence diagrams, namely those on Figure B.1, Figure B.3, and Figure B.4. The initialization of the supporting components as well as of the MonAgent and SelfOptAgent are both presented on Figure B.3 and Figure B.4.

In general, all bootstrapping interactions follow the principle of the local *Bootstrap Manager* starting the node FDH components in the right order, after which each of the components establishes a connection to the central FDH *Configuration Server*. This Configuration Server keeps, on one hand, the network-wide and, on the other hand, the node-specific FDH configurations. Thereby, each component downloads its belonging models, which facilitate its operation in the course of achieving distributed self-healing on top of the network to take care of. Besides downloading its configurations from the centralized server, each node level FDH component establishes communication channels to the other FDH node components it has to pass information to. The establishment of these communication channels poses constraints with respect to which components should be already running before a particular FDH module can be successfully started. Correspondingly, the constraints determine the order in which the FDH components are bootstrapping within a network node.

As already mentioned, each FDH component gets its specific configurations from the centralized FDH Configuration Server. In the following, the different configurations - policies and models - are pointed out, which determine the run-time operation of the FDH framework and are obtained from the FDH Configuration Server. The steps of obtaining these models and policies are visible on the sequence diagrams describing the bootstrap and initialization processes, i.e. Figure B.1, Figure B.3, and Figure B.4:

- The FDH node local SelfOptAgent downloads its node-specific optimization and action synchronization models (*getOptimizationAndActionSynchronizationModels* on Figure B.3)

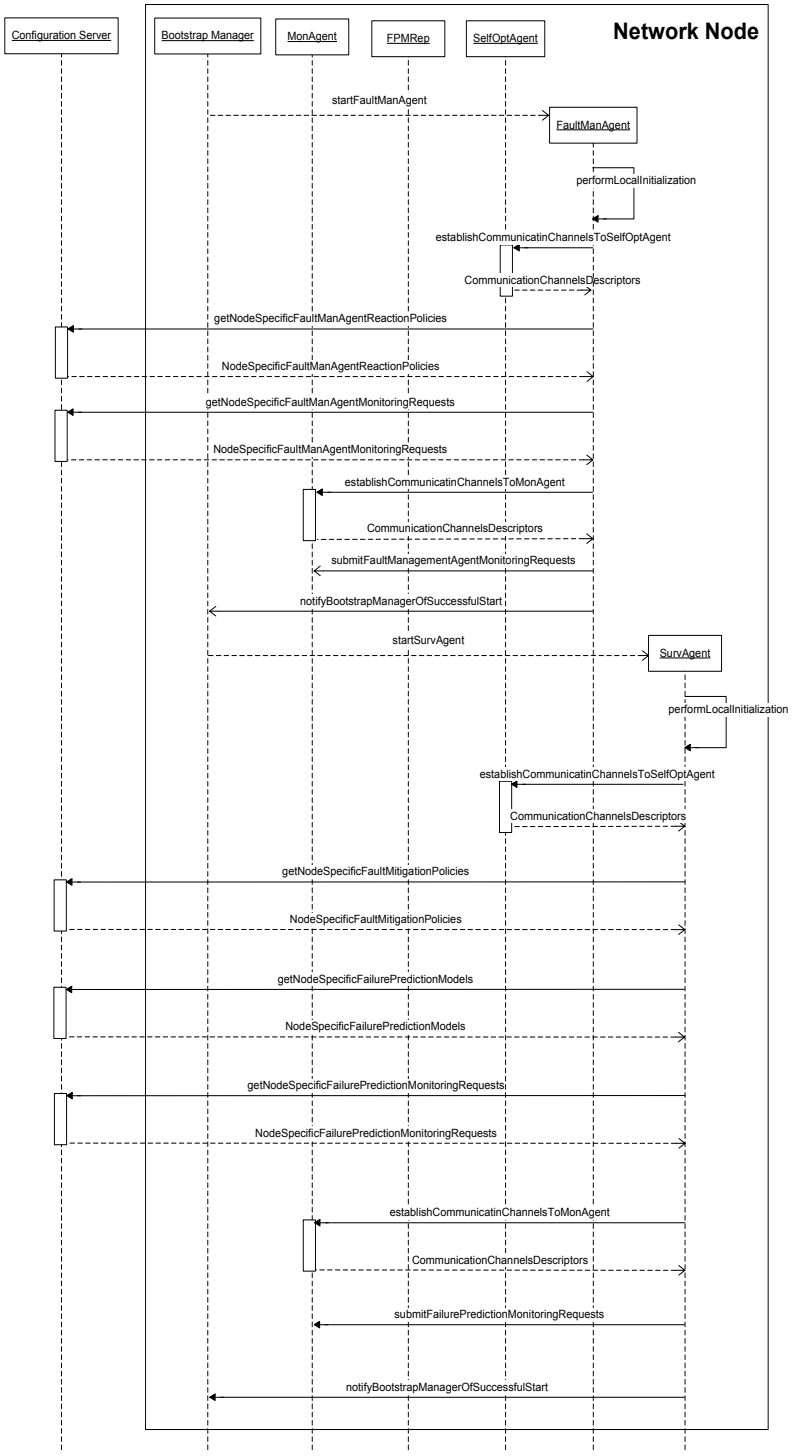

Figure B.1: FDH Initialization on Node Level: Bootstrapping the key Self-Healing Agents

from the Configuration Server.

- The Fault-Propagation Model Repository in each FDH node downloads the network-wide Fault-Propagation Model from the Configuration Server (*getNetworkWideFaultPropagationModel* on Figure B.3).
- Furthermore, each FDH MonAgent in a node obtains the different types of policies it requires - *Admission*, *Notification*, and *Adaptation Policies* - from the centralized FDH Configuration Server.
- The Event Dissemination Agent requires a list of IP addresses of FDH nodes, such that it can perform alarm/incident/event dissemination. These addresses are also downloaded from the centralized Configuration Server after the EvDissAgent has established a connection to it (*getListOfAddressesOfInvolvedNetworkNodes* on Figure B.4).
- Moreover, the policies that steer the *Asserted Incidents Assessment Functions* are also supplied by the network operations personnel to the FDH Configuration Server and downloaded (*getNodeSpecificAssIncAssFuncPolicies* on Figure B.4) by the corresponding node supporting components (i.e. *Node Incident Repositories* during the boot up of the local FDH instance.
- During the bootstrapping of the FaultManAgent and the SurvAgent, described in detail next and depicted on Figure B.1, following configurations are downloaded by the FaultManAgent from the Configuration Server:
  - Different types of reaction policies for the tasks of Fault-Isolation Assessment, Fault-Removal, and Fault-Removal Assessment - *getNodeSpecificFaultManAgentReactionPolicies* on Figure B.1
  - Monitoring requests to be submitted by the FaultManAgent to the MonAgent for the purposes of Fault-Detection - *getNodeSpecificFaultManAgentMonitoringRequests* on Figure B.1
- In addition, the SurvAgent downloads:
  - The node-specific Fault-Mitigation policies required in order to implement rapid immediate reactions to the symptoms of a faulty condition - *getNodeSpecificFaultMitigationPolicies* on Figure B.1
  - The node-specific Failure-Prediction Models for the FPF functions - *getNodeSpecificFailurePredictionModels* on Figure B.1
  - The node-specific monitoring requests required for the Failure-Prediction within the SurvAgent - *getNodeSpecificFailurePredictionMonitoringRequests* on Figure B.1.

Having described the general structure of the FDH bootstrapping procedure, the process of FDH initialization is next zoomed into from the perspective of the key self-healing agents, i.e. the FaultManAgent and the SurvAgent. As previously mentioned, the belonging interactions are visualized in Figure B.1. For the interactions on Figure B.1, it is presumed that the MonAgent, FPMRep and the SelfOptAgent are already started (according to the procedure described on Figure B.3).

The flow on Figure B.1 proceeds with the Bootstrap Manager starting the FaultManAgent (*startFaultManAgent*). Initially, the FaultManAgent performs its local initialization, which might include various aspects such as allocating required memory etc. Subsequently, the FaultManAgent

establishes a communication channel to the local SelfOptAgent, in order to access it for the purpose of action synchronization during the operation of the network. Next, a connection is established to the FDH Configuration Server and a set of reaction policies is downloaded for the different tasks of the FaultManAgent control loop (*getNodeSpecificFaultManAgentReactionPolicies*), as described above. These include node-specific policies for the *Fault-Isolation Assessment*, the *Fault-Removal*, and the *Fault-Removal Assessment* tasks of the local control loop. In addition, the monitoring requests are downloaded (*getNodeSpecificFaultManAgentMonitoringRequests*), which are used by the FaultManAgent to request the MonAgent to instrument the monitoring jobs required for local Fault-Detection (*submitFaultManagementAgentMonitoringRequests*). These requests are submitted after the belonging communication channels towards the MonAgent have been established. After all these steps have been conducted, the bootstrapping of the FaultManAgent within an FDH node is concluded and a corresponding notification sent to the Bootstrap Manager (*notifyBootstrapManagerOfSuccessfulStart*).

After the *FaultManAgent* was successfully started, the *Bootstrap Manager* initiates the start of the *SurvAgent* within an FDH node. The SurvAgent also performs its local initialization at first, and proceeds with establishing its communication channels to the local *SelfOptAgent*, for the purpose of action synchronization and optimization during the operation of the network. Subsequently, the SurvAgent connects to the *Configuration Server* and downloads its node-specific Fault-Mitigation Policies (*getNodeSpecificFaultMitigationPolicies* on Figure B.1), Failure Prediction Models (*getNodeSpecificFailurePredictionModels*), and monitoring requests (*getNodeSpecificFailurePredictionMonitoringRequests*) for the data required in the scope of the FPF module of the SurvAgent. Having downloaded all these configuration data/models from the FDH Configuration Server, the SurvAgent proceeds with establishing its required communication channels to the local MonAgent, and submitting the monitoring requests to support the process of Failure-Prediction (*submitFailurePredictionMonitoringRequests*). Finally, the Bootstrap Manager is notified of the successful start of the SurvAgent and proceeds with the interactions depicted on Figure B.4, which describes the starting of the key supporting components such as the *alarm/incident repositories* and the *Event Dissemination Agent*. The bootstrapping of these components follows a similar scheme as the one described in detail for the FaultManAgent and the SurvAgent.

The above bootstrapping description contains two calls, which trigger the instrumentation of monitoring services for the needs of Fault-Detection (*submitFaultManagementAgentMonitoringRequests*) and Failure-Prediction (*submitFailurePredictionMonitoringRequests*). These two calls stand for the process of monitoring request admission and the setting up of monitoring jobs for the needs of the FDH self-healing agents. Therefore, the following text elaborates on the MonAgent interaction sequences behind these two calls. Figure B.2 details the process of setting up a monitoring job within the FDH MonAgent. It starts with a client agent submitting a monitoring request (*submitMonitoringRequest*), which is in turn evaluated by the Orchestration Engine regarding whether it can be accepted for execution (*evaluateAdmissionPolicies*). This evaluation is determined by some admission policies that may concern security or performance aspects, e.g. whether there are enough memory resources to setup the job. In case the monitoring request cannot be satisfied, a reject type of message is sent back to the requester (*rejectMonitoringRequest*). In case the admission policies have allowed the request to pass, its instrumentation needs to be synchronized (*synchronizeTentativeAction*) over the SelfOptAgent with other potential tentative actions coming from the FDH agents. Given that the SelfOptAgent has allowed the setting up of the monitoring job, the Orchestration Engine has to recognize the type of request - traffic or node monitoring - and forward (*forwardTrafficMonitoringRequestToTrafficMonCoord* or *forwardNodeMonitoringRequestToNodeMonCoord*) it to the belonging monitoring coordination component,

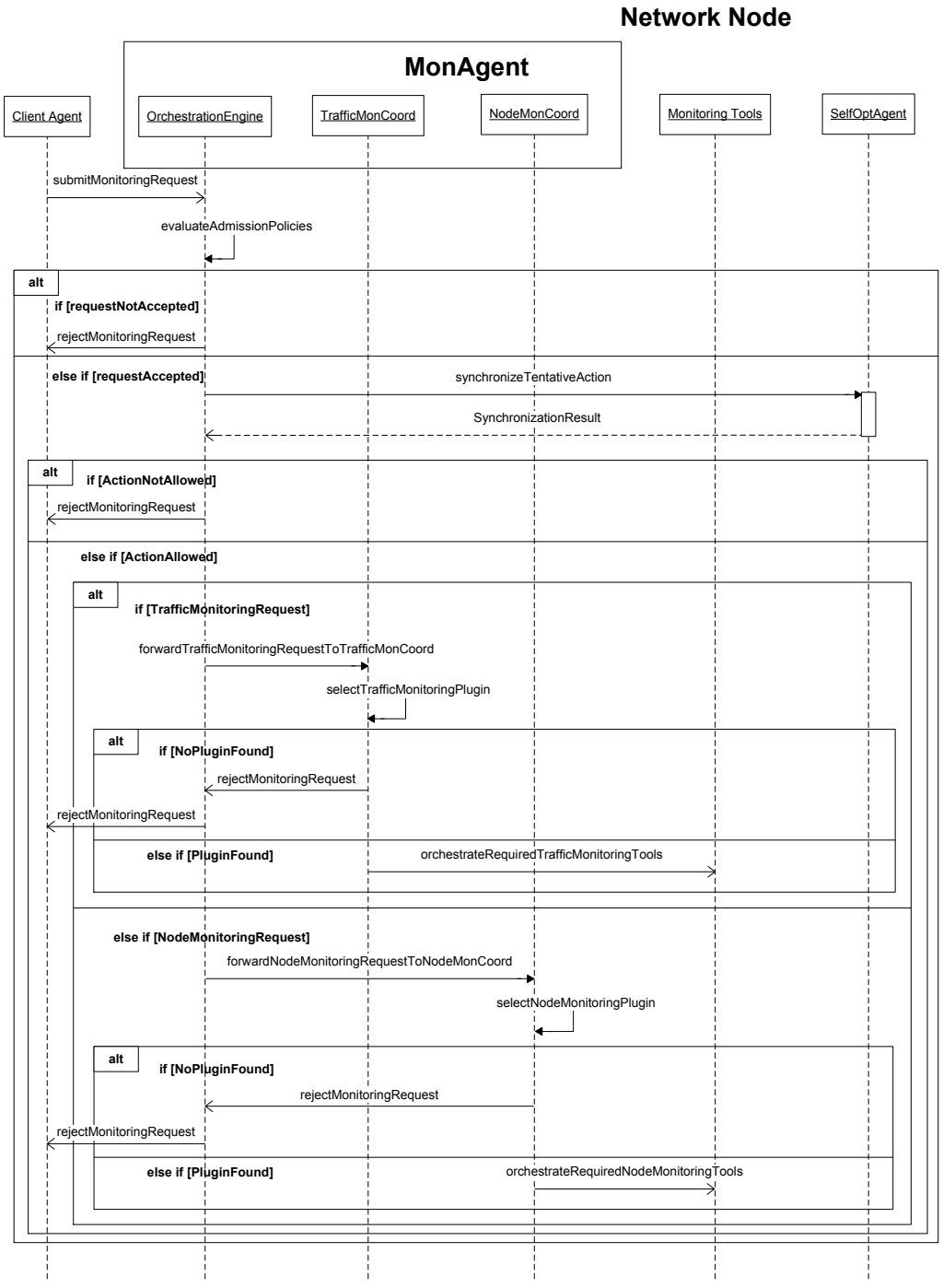

Figure B.2: Setting up a Monitoring Job

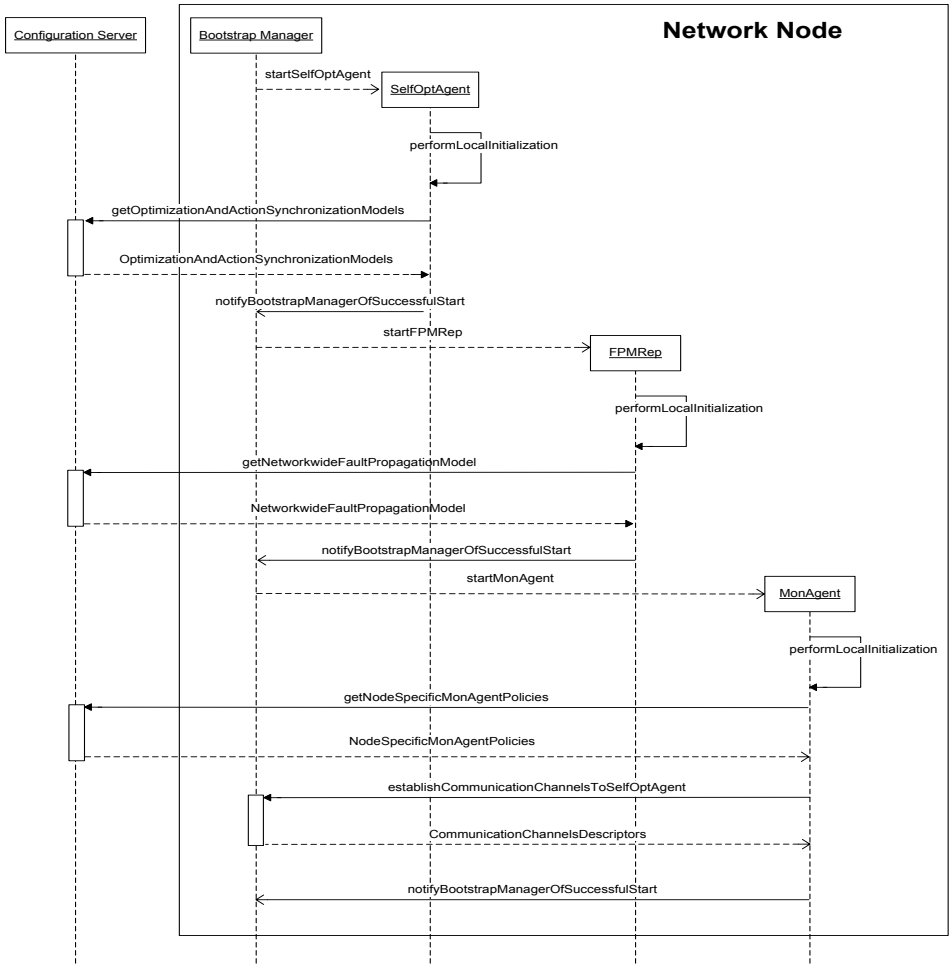

Figure B.3: FDH Initialization on Node Level: Bootstrapping of the SelfOptAgent, the MonAgent and the Fault-Propagation Model Repository

i.e. Traffic Monitoring Coordination (TMC) or Node Monitoring Coordination (NMC). The monitoring coordination component has in turn to scroll over the available plug-ins and select the one capable of processing the request - *selectTrafficMonitoringPlugin* or *selectNodeMonitoringPlugin*. In case no such plug-in is available, a reject type of message is communicated back to the requester. Finally, in case the right plug-in was identified, the monitoring job is setup and the belonging monitoring tools are used to observe relevant key performance indicators (*orchestrateRequiredTrafficMonitoringTools* or *orchestrateRequiredNodeMonitoringTools*).

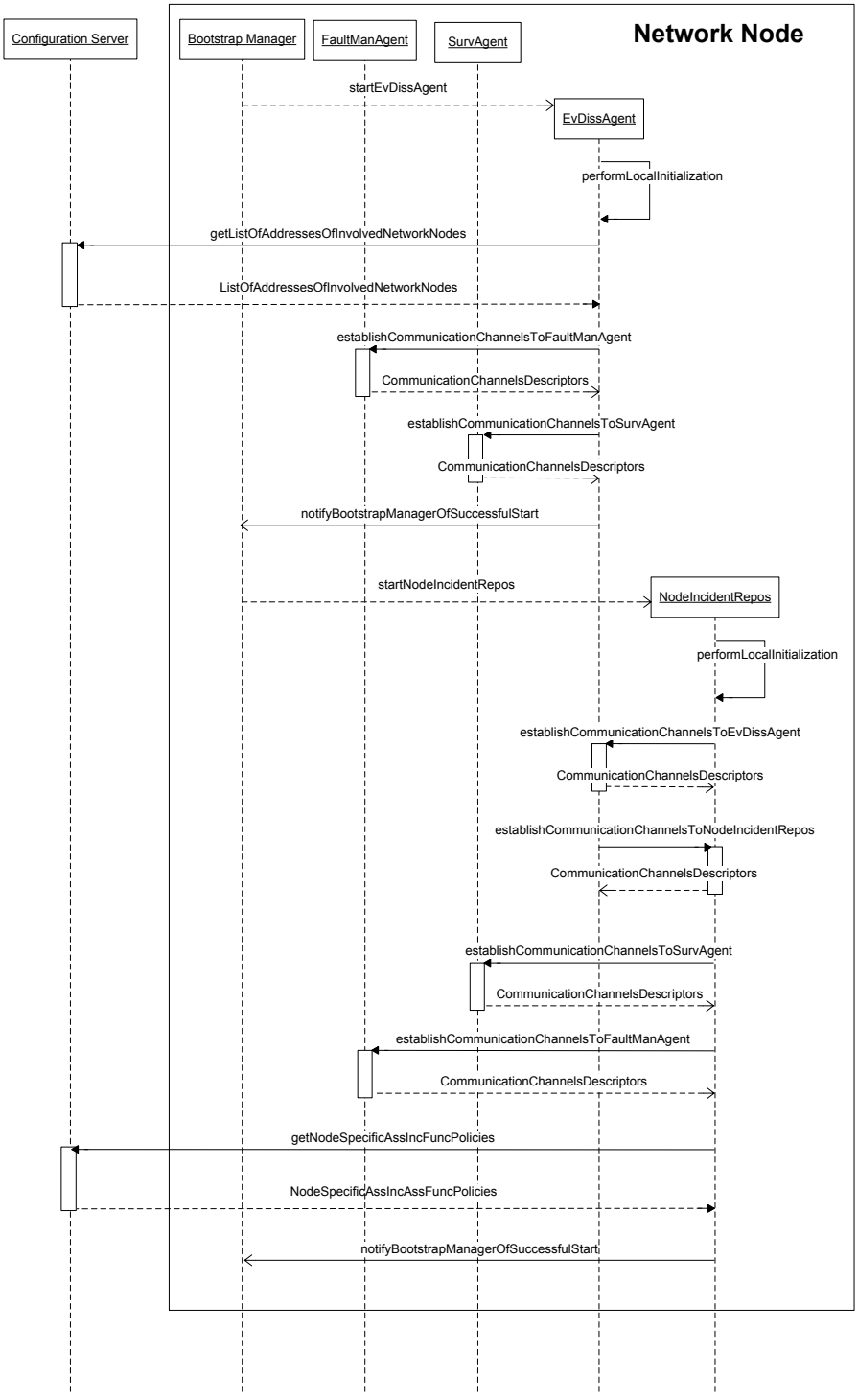

Figure B.4: FDH Initialization on Node Level: Bootstrapping of key Supporting Components

Appendix C

Examples of Faulty Conditions resolvable by FDH

In this section, some examples of concrete operational issues/problems are given, which can be resolved in case different access/core nodes and end systems implementing FDH collaborate. Some of these problems were used within the case studies - in section 13 as well as in the specific case studies in the chapters on various algorithms and/or components - to exemplify various aspects of the proposed FDH framework. The following list is meant to give an idea of problems where FDH can be extremely helpful in increasing the overall Quality of Service of the network/system in question. This list complements the general classification of network/system problems provided in section 5.3.

Black Holes: *Black Holes* in IPv6 networks, due to misconfiguration in the course of IPv4-to-IPv6 transitioning, constitute one of the key case studies in chapter 7, chapter 8 and chapter 13 in this thesis.The general term *Black Hole* - beyond the scope of IPv6 - represents a family of packet delivery problems where the physical (and in some cases even logical/routing) connectivity between two systems is present, but however the packets sent between the two nodes (e.g. hosts) don't reach their destination. The erroneous state results from the fact that the systems, even having the capabilities to react to the fault activation, do not get notified of packet delivery failures and can not even localize the fault. That is, the sender may continue sending packets without detecting the packet loss problem and cannot correspondingly react. The forwarding nodes are also not aware of the problem and do not adapt their behavior correspondingly. The phenomenon of Black Holes has been extensively studied and its relevance for ISP operators is now well known [KYGS07] [KBMJ+08]. Potentially, Black Holes may occur due to:

1. Loss of connection because of an incorrect PMTU or broken tunnels, e.g. MPLS
2. Software bugs
3. Delayed routing protocol convergence etc.

The FDH architecture can remediate some Black Hole issues by detecting and isolating them, reconfiguring the corresponding core network components, and informing the end systems in question to adapt their behaviors.

Duplex mismatches: IEEE 802.3 Ethernet networks constitute a widely used link layer communication technology for local area networks. In that context, the IEEE 802.3 defines an auto-

negotiation procedure between the network interfaces of the involved nodes, such that no or less manual configuration is needed when setting up a local area network. This is of great use, since local area networks are often managed and set up by non-experts or just users of ICT technology. The steps and elaborations, on how a duplex mismatch based on the auto-configuration procedure can emerge, are described in detail in the main part of this thesis (e.g. section 13.2). The manifestation of such a problem can range from duplicate ACKs in TCP flows, fluctuations in the CWND (congestion window) size of TCP connections, to increased frame collisions and losses within the network interfaces [SC05]. The majority of these manifestations are experienced on the end systems, e.g. CWND size fluctuations. In case, symptoms of a duplex mismatch are observed within the nodes on a network path, then the FDH framework would disseminate the information to the involved network components and facilitate a resilient reaction in order to sustain QoS.

A possible reaction (as implemented in the case studies from the main part of the thesis) might be to restart the mismatching NICs on the nodes in question, in order to re-trigger auto-negotiation. A short term reaction for mitigating the duplex mismatch problem would consist of triggering the end system, which are pushing traffic over the affected link, to hold back outgoing packets for a while, whilst the NICs in questioned have finished restarting and auto-negotiating the link layer settings. That way, traffic is not unnecessarily (since it will for sure get lost during the Fault-Removal phase) pushed into the network, and the user will probably experience just a short delay.Further details and concrete examples can be found in the case studies throughout the main part of the thesis, especially section 13.2.

(D)DoS detection: Obviously, the mechanisms of the FDH architecture can also be employed for security related issues realizing a self-protecting/defending functionality. Recently, a lot of research has been conducted in the area of intrusion and cyber attack detection. The study and classification of traffic anomalies towards the detection of malicious software preparing a Distributed Denial of Service (DDoS) attack is an ongoing research topic. The results of this advanced research can be easily integrated with the FDH incident sharing mechanisms. In this way, end systems that detect such suspicious activities may send this information to the FDH instance at the edge of the ISP's network. After having correlated these notifications, and having identified the threat of a DDoS attack, the FDH instance on the edge can implement a policy in order to counter the intended attack.

Corrupted services (e.g. DNS, OCSP ...): Given that an end system or any functional entity has detected a malfunctioning service, the FDH incident sharing mechanisms can be employed to disseminate this information and inform other potentially affected end systems. For instance, an end system may detect the unavailability of a DNS server and use the FDH mechanisms, in order to share this information and enable other end systems to switch to another DNS server, as long as the primary one is offline. Furthermore, the FDH mechanisms can identify the crashed DNS server and restart it correspondingly, thereby notifying the network operations personnel regarding the occurred situation and the undertaken actions.

Maintenance activities: According to [MIB+08], maintenance activities are responsible for around 20% of the failures observed in an operational IP backbone. The FDH architecture can potentially detect the corresponding outage, identify the reason for it and apply, for example, some admission control policies in order to guarantee best QoS for the already subscribed end systems.

PMTU issues: Traditionally, IP protocols implement a PMTU (Path Maximum Transmission Unit) discovery procedure on end systems. PMTU discovery is performed before the data transmission has started. This enables the end systems to obtain a valid PMTU for a connection towards another end system using the services of an operator's network. However, it is possible that the PMTU

towards a host decreases during the lifetime of a connection. In such cases, the router at which the PMTU has decreased will fail to forward a packet and is expected to respond to the end system with an ICMP message indicating the packet loss and the reason. The IP module on the sender end system is expected in turn to readjust its PMTU settings towards the receiver. However, this procedure is very often not possible due to reasons such as firewall ICMP suppression on routers - as in the case of the IPv6 Black Holes used for the case studies - or simply because the traffic that cannot be forwarded is being tunneled (e.g. a VPN tunnel) and the ICMP messages are correspondingly not relayed. In such cases the sending host fails to adapt its fragmenting behavior and the packets towards the receiver fail to reach their destination eventhough there is an intact physical connectivity. Obviously, this constitutes a specific type of a Black Hole as previously described. In that context, a framework such as FDH can enable the implementation of mechanisms which allow the collaborating end systems, access and network components to overcome such issues with sudden PMTU changes.

Appendix D

Bayesian Networks

The elucidations below represent a mathematical definition of the Bayesian Network framework. Thereby, the following items constitute the main mathematical components for defining the belonging inference problem:

- n - number of random variables/nodes in the DAG
- $X_i : \Omega_i \rightarrow D_i$, with $i \in \{1, \ldots, n\}$ - random variables where D_i is a finite set. The tuple (Ω_i, F_i, P) denotes a probability space, where Ω_i is the sure event, F_i is the σ-field/algebra of events, and P is the belonging probability measure.
- $V := \{X_1, X_2, \ldots, X_n\}$ - set of random variables/nodes in the DAG. Usually, they represent states of network objects/components/entities (in the case of a Dependency Graph [SS04a]) or the occurrence of network events (in case of a Causality Graph [SS04a]).
- $E := \{e_{ij}\}$, $i, j \in \{1, \ldots, n\}$ - set of edges in the DAG. They usually represent cause-effect relationships between two connected nodes X_i and X_j (e.g. X_i causes X_j).
- $Pa(X_i) := \{X_{i1}, X_{i1}, \ldots, X_{il}\}$, $i \in \{1, \ldots, n\}$, $l < n$, $X_{ij} \in V$ - set of parent nodes for X_i, i.e. nodes which influence/cause X_i.
- M_i – (| $Pa(X_i)$ | +1)-dimensional conditional probabilistic matrix of size | D_i | × | D_{i1} | × . . . × | D_{il} | associated with the random variable X_i. The component at position $(x_i, x_{i1}, x_{i2}, \ldots, x_{il})$ of the matrix M_i is given by [SS04b]: $M_i(x_i, x_{i1}, x_{i2}, \ldots, x_{il}) = P(X_i = x_i \mid X_{i1} = x_{i1}, X_{i2} = x_{i2}, \ldots, X_{il} = x_{il})$, $x_{ij} \in D_{ij}$, $j \in \{1, \ldots, l\}$, $l < n$, with D_{ij} a set of possible values taken by X_{ij}.

A Bayesian network can then be precisely defined as a pair (G, P), where $G = (V, E)$ is a DAG satisfying the Markov condition, which means that the probabilities associated to each node of G are influenced only by the direct neighbors/predecessors of the node. Hence, each variable $X \in V$ is conditionally independent of the set of all its non-direct neighbors, given the set of all its parents (i.e. direct predecessors in the DAG) [Nea03], i.e.

$$P = (X_1 = x_1, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i \mid X_j = x_j, \forall\, X_j \in Pa(X_i)) \tag{D.1}$$

Furthermore, denote by $C \subset \{1, \ldots, n\}$ the index set of random variables with known/observed sample values and by $o_j \in D_j$, $j \in C$, the known/observed value taken by the random variable X_j. A Fault-Localization problem can be seen as the problem of finding the most probable explanation of the observed symptoms/alarms based on the Belief Network used as a fault-propagation model [Nea03]:

$$A^* = \underset{A^* \in S}{\operatorname{argmax}} P(X_1 = a_1, \ldots, X_n = a_n) \tag{D.2}$$

with $S = \{(a_1, \ldots, a_n) \in D_1 \times \ldots \times D_n \mid a_j = o_j, \forall\, j \in C\}$. This problem is obviously NP-hard and despite the existence of polynomial algorithms for some specific types of belief networks - such as singly connected networks [SS04a] - the algorithmic solution of the above formulated Fault-Isolation problem is a difficult task in the scope of the envisioned automated self-healing.

Appendix E

Example of an Adaptive Monitoring Behavior

Given the FDH monitoring framework, described in section 6.2.1.1 and section 7.1, the following pragraphs provide an example of an adaptive monitoring behavior that can be realized by means of the Adaptation Policies of the FDH MonAgent.

E.1 Monitoring QoS Violations

Monitoring QoS violations is an important aspect for network operators in the course of fulfilling contractual SLA agreements. Vital QoS metrics include jitter (delay variation), round trip time (RTT), throughput, and bandwidth among the edge routers of a service provider network. Hence, during the FDH MonAgent design and development, it was experimented with mechanisms for monitoring these metrics and adapting the belonging monitoring behaviors, processes and correspondingly plug-ins of the FDH MonAgent.

For the purpose of monitoring the above QoS metrics, an FDH MonAgent instance was deployed on the edge routers of the network in Figure 11.1, i.e. on R1, R4, R8, R3, R7, and R11. Two plug-ins were developed in order to measure the aforementioned metrics between any combination of ingress (R4 and R8) and egress router (R1, R3, R7, and R11). The first plug-in wraps the *ping* tool for measuring the round trip time between the ingress-egress pairs. The second one wraps the *iperf* tool - client component on the ingress routers and server component on the egress side - in order to actively measure the rest of the metrics - jitter, throughput, and bandwidth for UDP and TCP connections, as well as out of order packets. In the following, the *ping* - wrapping plug-in is denoted as RTT plug-in, and the *iperf* - wrapping one as QoS plug-in[1].

An exciting part of this setup is given by the efforts to develop adaptation strategies for the RTT plug-in. Similar efforts are given by [HCG01] and [AAFR10]. In general, the adaptation of different monitoring behaviors is a large research area which is not the focus of this thesis. In that context, the example given here is mainly meant to illustrate the potential for implementing such adaptation behaviors within the FDH MonAgent framework.

Based on RTT data recorded in real service provider networks,a simple controller was developed that adapts its sampling rate dynamically in an effort to reduce the amount of active pings. The

[1] These plug-ins were also used as a basis for various discussions and measurements in section 7.1.3 and chapter 13

main idea is to record some RTT measurements and to use them as "trainings data" for obtaining an interval within which some significant changes in the RTT dynamics are expected. Moreover, the data monitored within these intervals is every time evaluated regarding whether it really contains some significant dynamics. If none, then some new "trainings data" is obtained and a new interval calculated. The controller works as follows in a summarized form:

Input:
1) N - number of training measurements.
2) $clSize$ - cluster size for the significant changes in the training data
3) tol - determines whether the difference between measurements is significant or not
4) α- confidence level for constructing a confidence interval with the *t-distribution* [Sch12]
Preconditions: $N > 0$, $clSize > 0$, $tol > 0$, $0 < \alpha < 1$

1. **Record** *N* RTT values
2. **Obtain** clusters of significant changes (based on *tol*), within the recorded values, according to a pre-specified cluster size *clSize*
3. **Calculate** a mean value for the distance between significant changes in the RTT dynamics based on the center positions of the clusters
4. **Calculate** a confidence interval of level α around the mean value using the *t-distribution* [Sch12]
5. **Monitor** RTT values between the points in time suggested by the lower and upper bounds of the confidence interval
6. **Conduct** a test on the monitored RTT values
   → If no significant change found, go to (1)
   → If any significant changes found, go to (5)

Obviously, such an adaptation strategy is easily implementable within the adaptation policy hook of the FDH MonAgent prototype. As previously mentioned, the strategy was used on RTT data obtained from different service provider networks, in order to judge on its applicability. Within a MATLAB simulation, it turned out that this strategy is a good choice in case the RTT values are stable with periodical bursts of significant changes. In that case, it was possible to reduce the number of measurements thereby still being able to reasonably reconstruct the RTT signal dynamics. This is illustrated in Figure E.1 for *N = 100*, *clSize = 20*, *tol=20*, and α*= 0.6*. Thereby, a minimal mean error between the actual data and the signal extrapolated from the observed data was achieved (*0.7*), and the monitored values were reduced by *10%*. In overall, a training session - step (1) from the algorithmic description- was initiated only once in that setup. On the other hand, this technique performed poor in cases of continuous dynamics, where the confidence intervals were not really bringing any benefit with respect to reduction of observed data and the capability to reconstruct the RTT signal from the observed data.

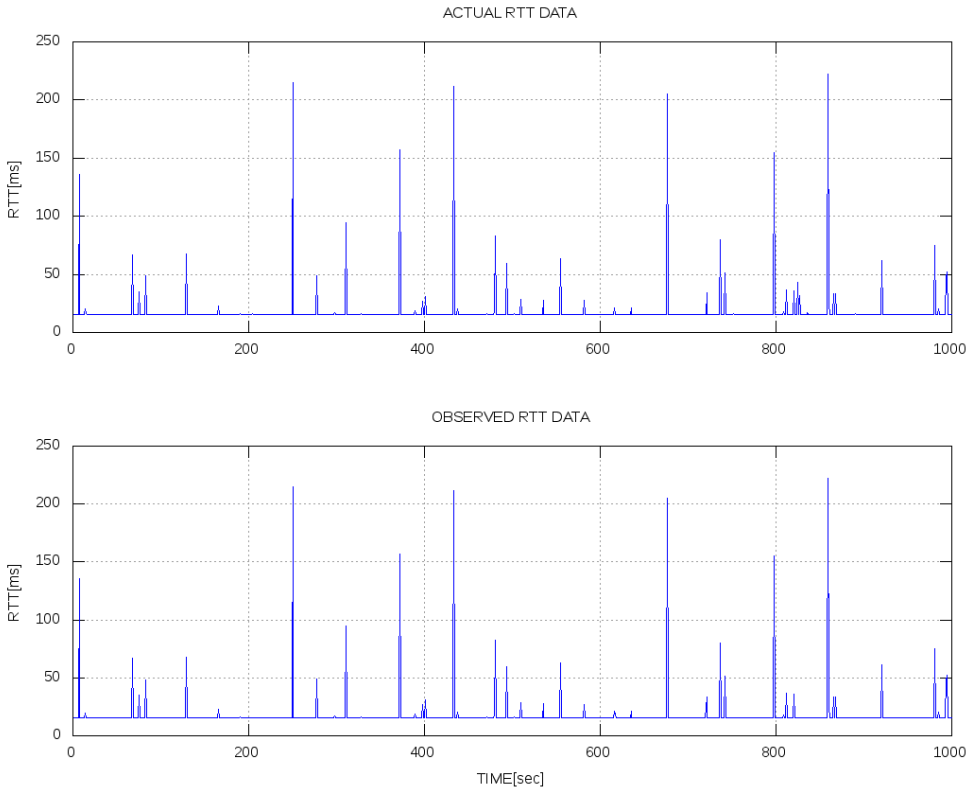

Figure E.1: Actual RTT data and observed RTT data with Adaptation

Appendix F

FDH Collaboration between ISPs and End Systems

In this section, the different FDH aspects of collaboration among end systems, access network and core network components are presented [1]. These aspects should be considered in implementing self-healing based on collaboration of FDH instances across multiple network segments. Furthermore, situations and issues are identified where the collaboration can be beneficial for enabling self-healing and fault resolution in the long-term operation of the involved networks. Thereby, it is assumed that the devices across the different networks are equipped with FDH components. Correspondingly, the collaboration aspects are specified as interactions between the FDH instances. The presented specifications and flows are described on an abstract level and can be implemented on top of various types of access technologies (e.g. DSL, 3GPP, UTRAN, or EPC).

F.1 Means for Auto-Collaboration

In general, the base for FDH collaboration over the domains in question is provided by the exchange of different types of information, either during the device/terminal subscription phase for the end systems, or during the phase in which the end users' equipment is utilizing the ISP's network. The information used by an FDH instance can be classified as follows:

1. run-time information about detected incidents - stored in the incident repositories of an FDH implementing device
2. Fault-Propagation Model that describes information about potential chains of events (fault → error ... →failure) - stored in the FPMR repository of a node
3. policy configurations for the FRF, FIAF, and the FRAF modules within the FaultManAgent
4. policy configurations for the FMF module of the SurvAgent
5. monitoring requests for Fault-Detection (FDH FaultManAgent in the nodes) and Failure-Prediction (FDH SurvAgent in a node) and
6. action synchronization and optimization models for the SelfOptAgent

[1] The work presented in this chapter was initially published in [TC10b].

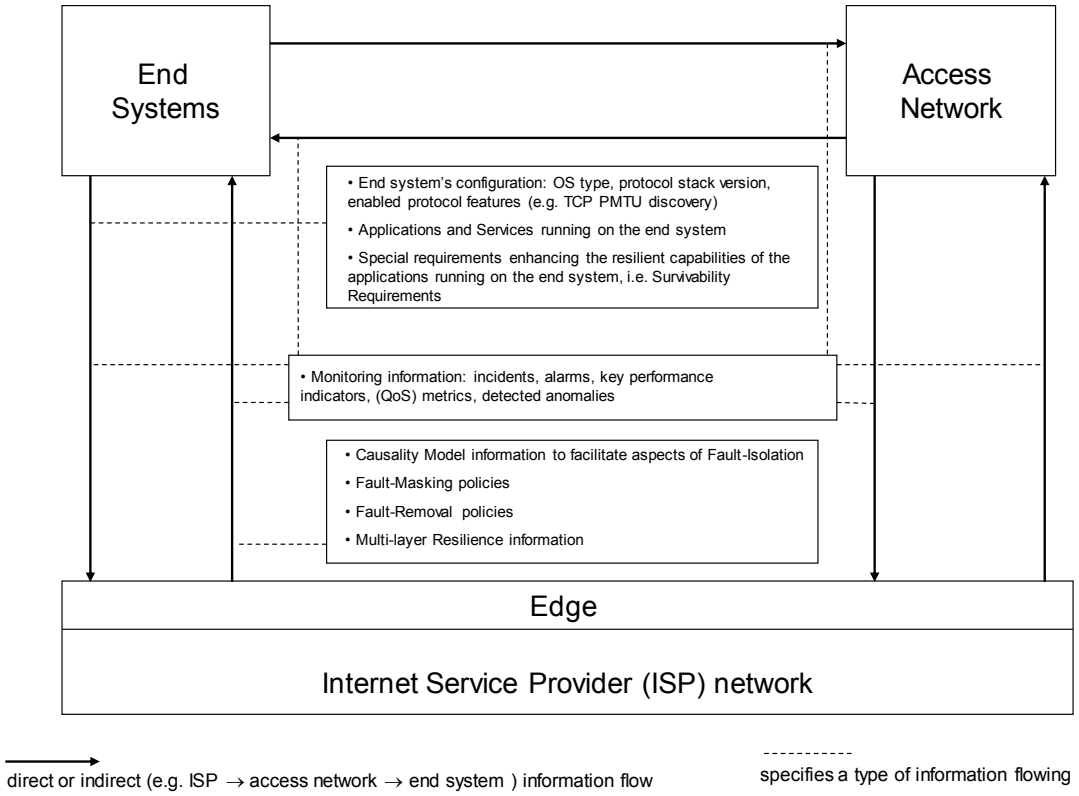

Figure F.1: Aspects of FDH Collaboration for Distributed Self-Healing

Figure F.1 gives an overview of the information flows required to facilitate collaborative self-healing. In the following subsections, the importance of the different types of information - outlined in Figure F.1 - is discussed - thereby distinguishing between the subscription phase of a device to the ISP network, and the operating phase when the network is utilized by the end system for accessing services and applications.

F.2 Auto-Configuration: Enabling FDH based Self-Healing across Multiple Network Segments

Starting with the subscription phase of end systems and devices: The *end systems* are expected to share - with the operator's network - information about their settings and configurations, e.g. OS (Operating System) type and characteristics, type and version of the employed protocol stack - including enabled protocol features (e.g. PMTU discovery), as well as information regarding applications and services hosted on the end system. This information - denote it *device capabilities* - is provided during the subscription phase of the device into the operator's network. The device capabilities can be seen as accumulated capabilities of the plethora of software and hardware components of an end system. As described in Figure F.2, the capabilities are required by the ISP network in order to select the appropriate FDH related configurations, which are sent back to the end system. This configuration/information may include (as illustrated in Figure F.1) a Fault-Propagation Model related to potential problems, which can occur in the network, such that the end system can better understand certain anomalies, furthermore Fault-Removal/Mitigation, Fault-Isolation Assessment and Fault-Removal Assessment policies, which enhance the set of such policies already in place on the end-system, as well as monitoring requests and action optimization models for the SelfOptAgent on the end system. An interesting issue is that of the information model that facilitates the processes in Figure F.2. The current trends in telecommunications are striving to reduce the usage of proprietary models and promote the usage of a single standardized model such as or CIM [CIM16]. For the purpose of self-description by end systems, one has to be aware that due to different standards and technological domains, the exchanged information will suffer major drawbacks because of the lack of a standardized terminology. Hence, the usage of semantic based model concepts such as ontologies is imperative for the process of self-description and auto-configuration for FDH. That is, the end system must submit (to the network) its capabilities in an ontological format such that the obstacle of different terminologies across software/hardware vendors, standardization bodies, etc. can be avoided.

Another aspect that must be addressed when specifying the Fault-Propagation Model as well as the various types of policies to be supplied to an end system, is that of confidentiality of the exported information. If not designed properly, the aforementioned models and policies may contain information that directly or indirectly reveals some sensitive information about the operator's network. Such information can be easily exploited by hackers. Therefore, it may be useful to use only cryptographically generated strings (i.e. avoid using full descriptions of the events) in the version of the Fault- Propagation Model which is given to an end system, or to delegate as many as possible reactions (e.g. Fault-Mitigation/ Removal) of the control loops to the edge or access network components.

Additionally, after getting to know the capabilities of the new subscriber, the FDH mechanisms of the network have to prepare the information flows from the network to the subscriber such that the end system can access information about incidents in the ISP network, and can correspondingly react according to the aforementioned Fault-Mitigation policies. This means that the FDH op-

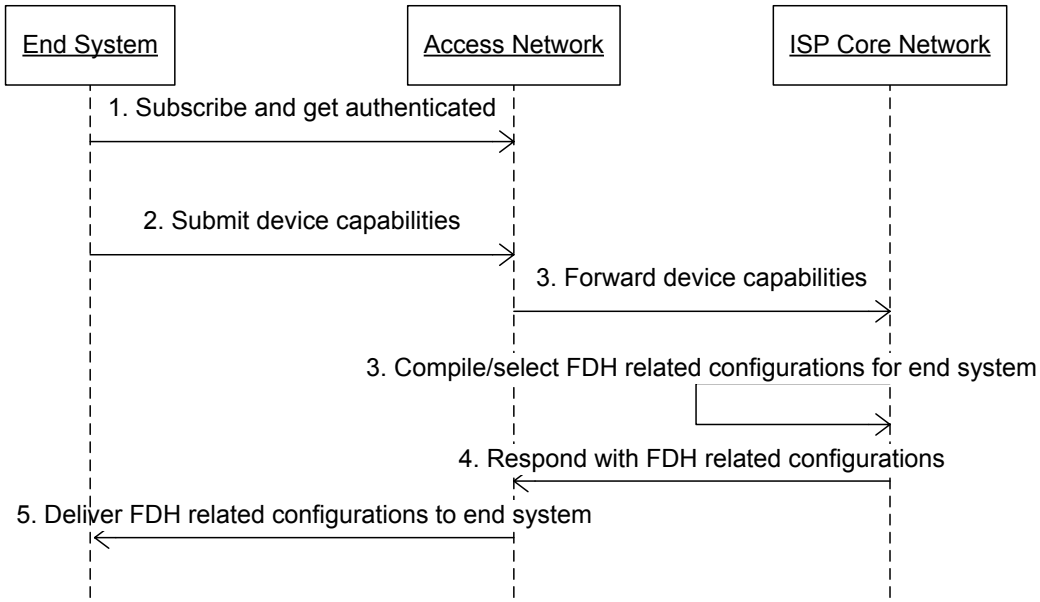

Figure F.2: FDH Auto-configuration of an End System

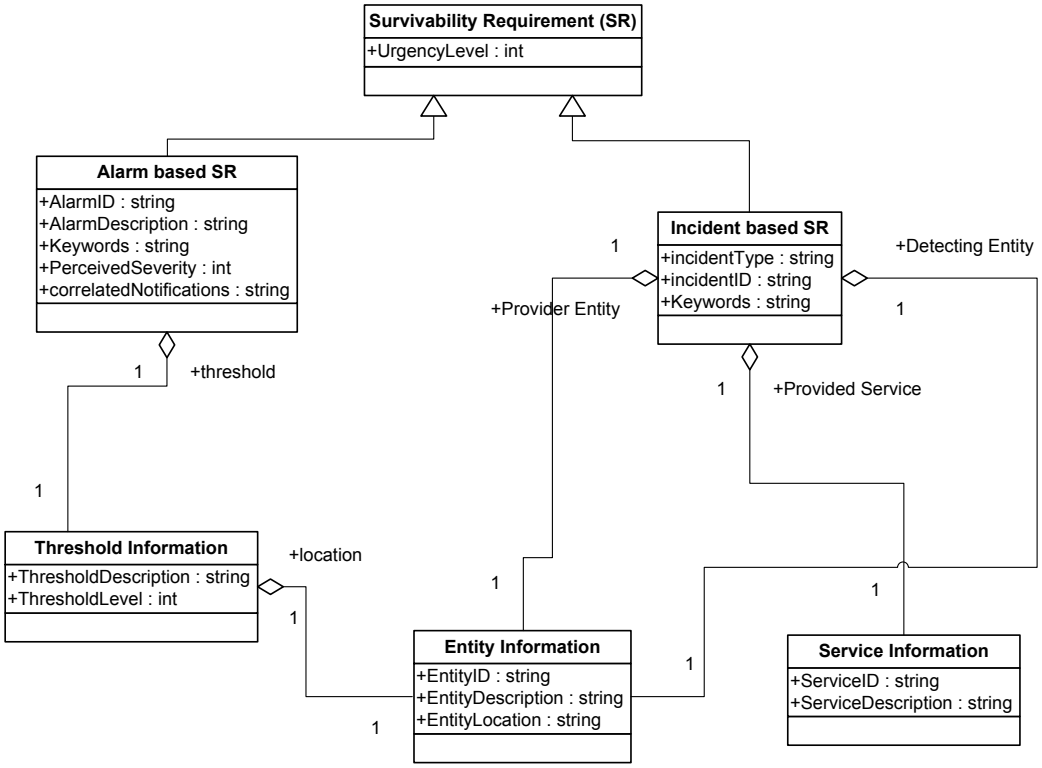

Figure F.3: Information Model for Survivability Requirements

erator's network mechanisms formulate *Survivability Requirements (SR)* for the applications and services on the newly subscribed end system. These SRs are then used as *filters* associated with the *FDH Node Repositories* and *EvDissAgents*, such that particular incidents and alarms (according to the SRs) are sent to the subscriber end systems. A similar concept of a Survivability Requirement was proposed by [Cha09], however without being elaborated in detail.

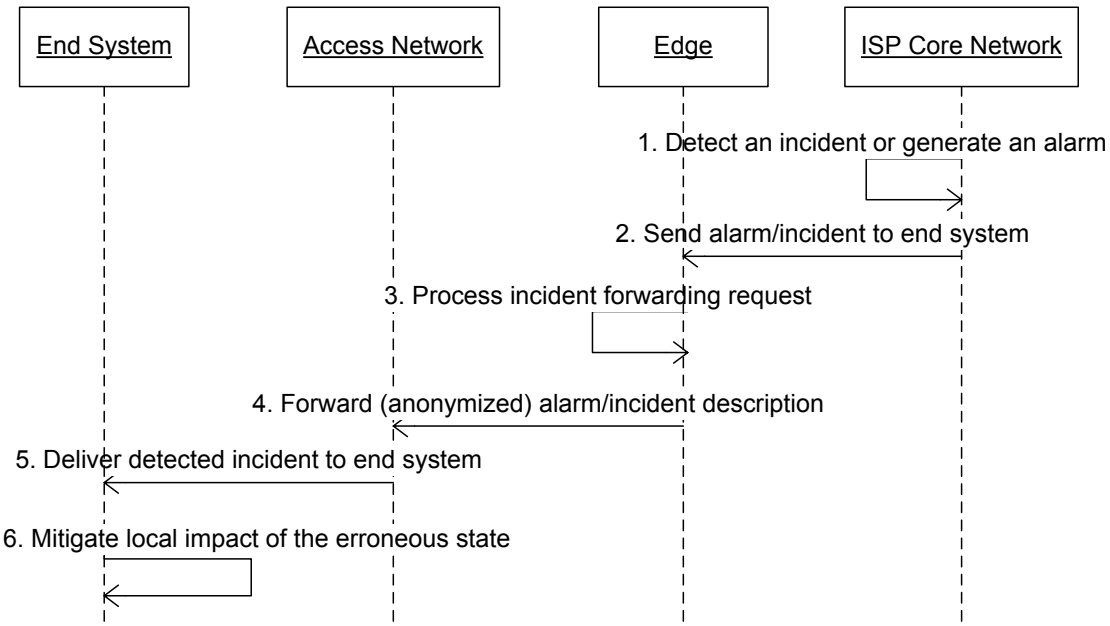

Figure F.4: An FDH Implementing End System Mitigating the Local Impact of an Erroneous State in the ISP Network

Figure F.3 depicts the proposed meta-model for a Survivability Requirement. This meta-model is based on alarm/incident event descriptions as recommended by ITU-T [ITU92] and CIM [CIM16]. According to the meta-model on Figure F.3, a Survivability Requirement has a particular *urgency level*, which might be used for prioritizing the dissemination of certain type of incidents/alarms over others. Furthermore, an SR can be of two types: 1) an incident based SR, or 2) an alarm based SR. The alarm type is described by some standard fields such as *AlarmID*, *AlarmDescription*, etc. In addition, the alarm type might link to a specific KPI which has crossed or is close to crossing a particular threshold. The threshold itself relates once again to a functional entity, e.g. a protocol module such as IP forwarding or an OSPF daemon inside a router. Moreover, an incident based SR is again described by some typical fields such as *incidentType*, *incidentID*, and *Keywords*, whilst at the same time referring to a particular networking service, which is fulfilled by a functional entity. In addition, it is possible to give some characteristics of the entity which is expected to detect the incident that this SR refers to. Based on such SR models, the network side FDH instances would be able to know which types of alarms/incidents must be sent to the subscribing end systems.

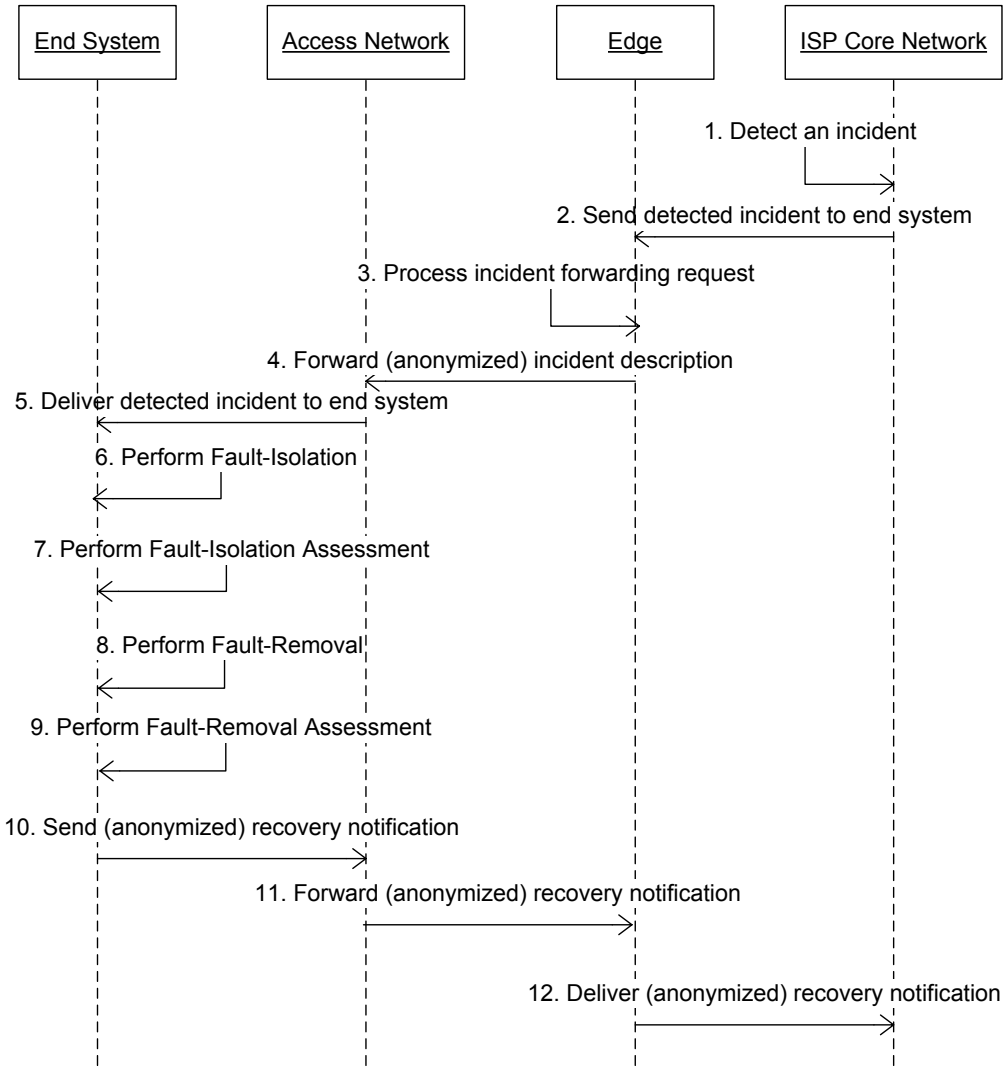

Figure F.5: An End System Performing Fault-Isolation and Fault-Removal in Case the Incident, detected in the ISP Network, indicates a Fault (Root Cause) on the End System

F.3 Collaboration and Information Flow during the Operation Phase

After an end system has subscribed to the ISP over the access network, selected types of monitoring information are expected to flow between the end system, the access network and the ISP's network. Figure F.4 describes a generic scheme of how an end system can mask the local impact of an erroneous state that has been detected in the core network (step 1). The node in the core network, on which the erroneous state was detected, reasons about the need to send the incident description to the corresponding end system, i.e. the decision would be based on a previously expressed/generated Survivability Requirement and/or on the end system being one of the end points in a flow in which some traffic anomalies were detected. Moreover, the Edge of the ISP network intercepts (step 3) the message (after it has been sent in step 2), and checks whether the information inside can be sent to the end system based on the security policies in place - e.g. in regard to privacy or confidentiality constraints. Obviously, such an incident message can contain information, which the operator does not want to share, in order to keep certain structures and configurations of her network a secret. Hence, the *Edge* should be equipped with a policy that allows/disallows, or manipulates alarm/incident descriptions being sent to end systems. Given that the message has been allowed or anonymized, step 4 and 5 deliver it to the end system. Based on the Fault-Mitigation policy, which has been supplied during the subscription phase of the end system (Figure F.2), the belonging FDH instance on the end system is expected to react and mask the local impact of the erroneous state indicated by the received alarm/incident description.

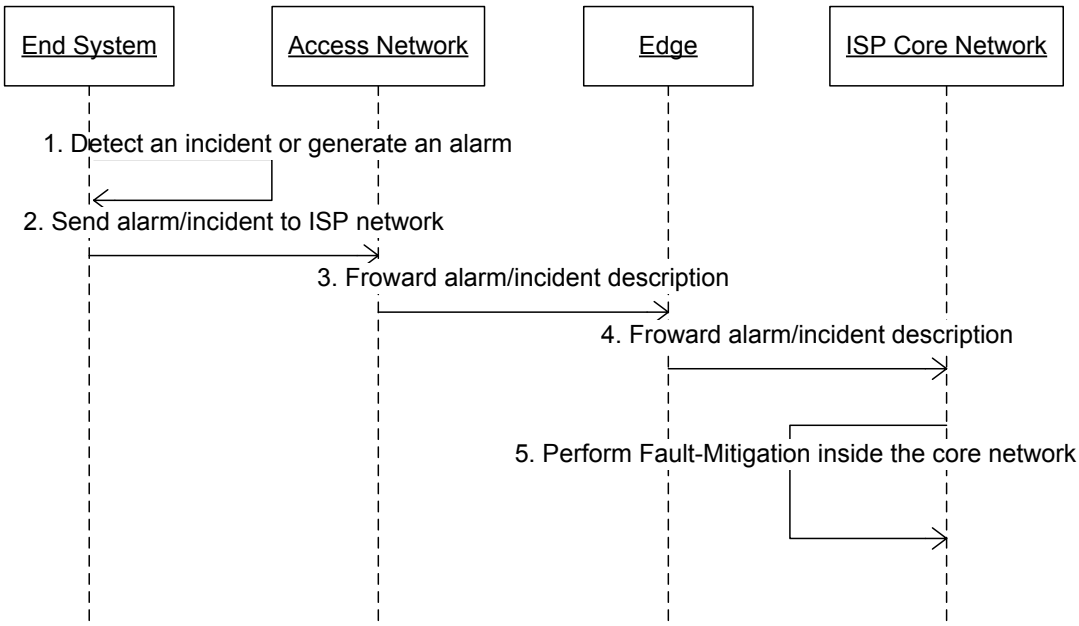

Figure F.6: Fault-Mitigation in the ISP Network based on Information Supplied by an End System

In addition to the mitigation behavior of the end system, which is realized by the SurvAgent of FDH, a behavior as the one described in Figure F.5 is also required. In this case the end-system *FaultManAgent* gets activated and performs Fault-Isolation (step 6), followed by the assessment of the Fault-Isolation result (step 7), and consequently Fault-Removal (step 8) in case the root cause for the traffic anomaly observed in the core network is located on the end-system. Finally, the removal of the fault is locally verified (step 9), and a recovery notification is sent back to the

ISP's network [2]. The logistic that facilitates these processes includes the Fault-Propagation Model as well as all the FaultManAgent policies - for Fault-Isolation Assessment, Fault-Removal and Fault-Removal Assessment - that must have been supplied during the subscription phase of the end system - see Figure F.2.

An additional aspect is given by monitoring information, i.e. metric values or incident descriptions of observed anomalies, which may flow from the end systems to the ISP core network, thereby enabling self-healing in the access network, the edge and the ISP core network. The signaling that realizes fault tolerant behaviors in the ISP core network is described in Figure F.6. In the course of this, an alarm/incident is conveyed from an end system to the relevant nodes in the ISP's network (steps 1, 2, 3 and 4). Subsequently, the relevant nodes in the ISP network perform Fault-Mitigation in order to handle the immediate local impact of the communicated symptoms.

In addition to the FDH behavior on Figure F.6, the sequence of interactions and actions towards eliminating the root cause(s) of a faulty condition in the core network are illustrated on Figure F.7. Again, the incident detected on an end system (step 1) is conveyed to the FDH instances in the ISP's network (steps 2, 3 and 4 on Figure F.7 where the relevant nodes (and belonging FDH instances) perform the tasks of Fault-Isolation (step 5 on Figure F.7, Fault- Isolation Assessment (step 6), Fault-Removal (step 7) and Fault-Removal Assessment (step 8). As in the previous descriptions, only the "straight forward" case when all steps are accomplished successfully is presented for illustrative purposes. The way various exceptions are handled in the course of achieving fault resolution is presented on the sequence diagrams in section 6.3.

Coming back to the interaction flows on Figure F.7: Presuming that a fault has been successfully removed in the ISP network, a recovery notification is disseminated to all the routers as wells as to the relevant end system subscribers (steps 10, 11 and 12 on Figure F.7). Thereby, the recovery notification would require to be anonymized/processed (step 10) according to confidentiality constraints such that sensitive information, e.g. regarding the topology of the operator's network, is not sent outside the belonging operator's domain.

Finally, as illustrated on Figure F.8, the sharing of incident event descriptions among end systems can enable FDH based resilient behaviors on the subscribed end user equipment. This sharing can be realized over the SOHO (Small Office Home Office) network or directly over the communication medium of the access network. It is also possible that different access network components (e.g. different "Node B"s and Remote Network Controllers in the context of UMTS), serving different subscribers, collaborate and enable the exchange of incident information between the subscribed end systems. An example of such interactions, facilitated by the FDH incident sharing mechanisms among end systems, is given by the possible realization of self-protection mechanisms in case of a distributed attack (i.e. DDoS), where multiple FDH instances on the end systems could exchange information related to the distributed preparation of a DDoS, and would synchronize in order to undertake countermeasures.

The collaboration aspects described in this chapter would enable FDH instances deployed across different domains - especially on end systems, different types of access networks, and ISP core networks - to exchange information required for the efficient management of alarms and incidents towards the realization of distributed self-healing. The above presented sequence diagrams capture the high level interactions required for the realization of Fault-Removal and Fault- Mitigation

[2] For illustrative purposes, the sequence diagrams are kept abstract without going into all the special cases, e.g. in case the Fault-Isolation result was not correct etc. Hence, the diagrams show only the "straight forward" cases when all the undertaken operations are considered successful. Detailed descriptions with possible exceptions to the flows are presented in section 6.3.

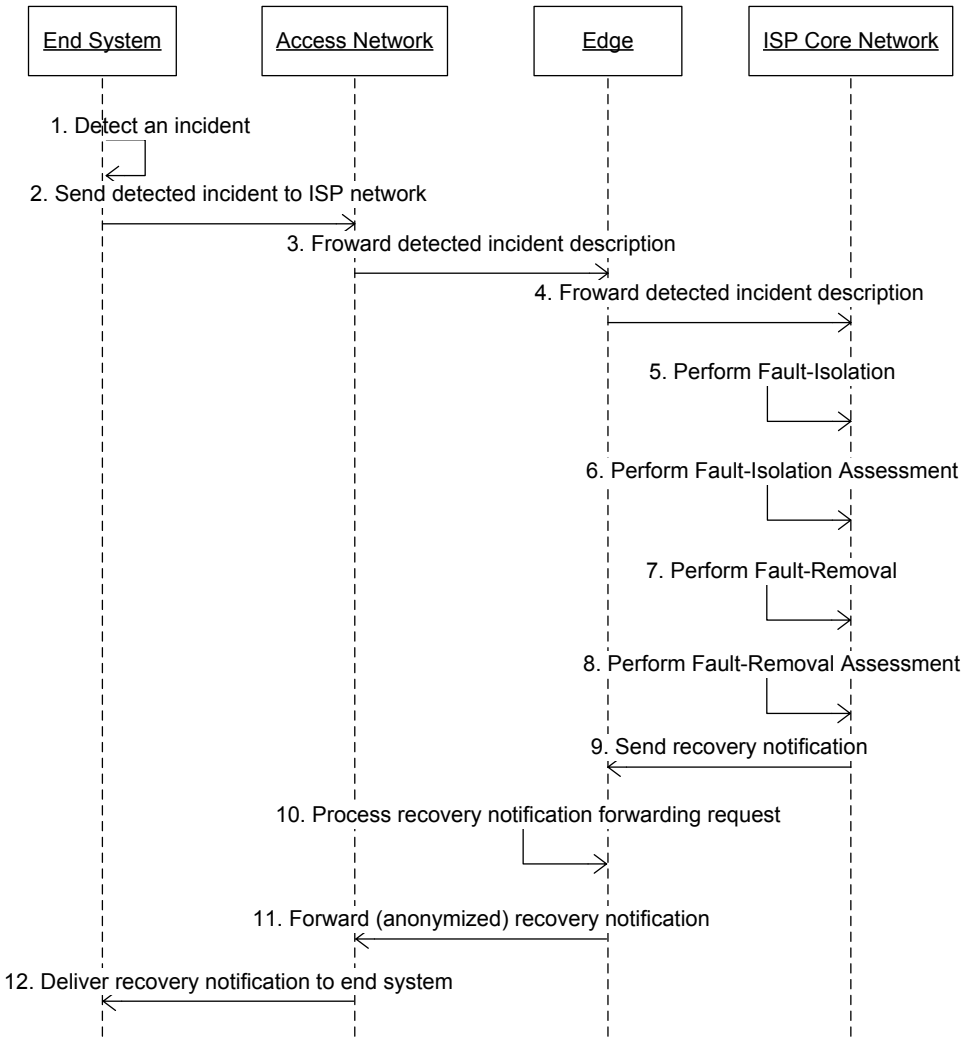

Figure F.7: Removing a Fault in the ISP Network based on Incident Detection on an End System

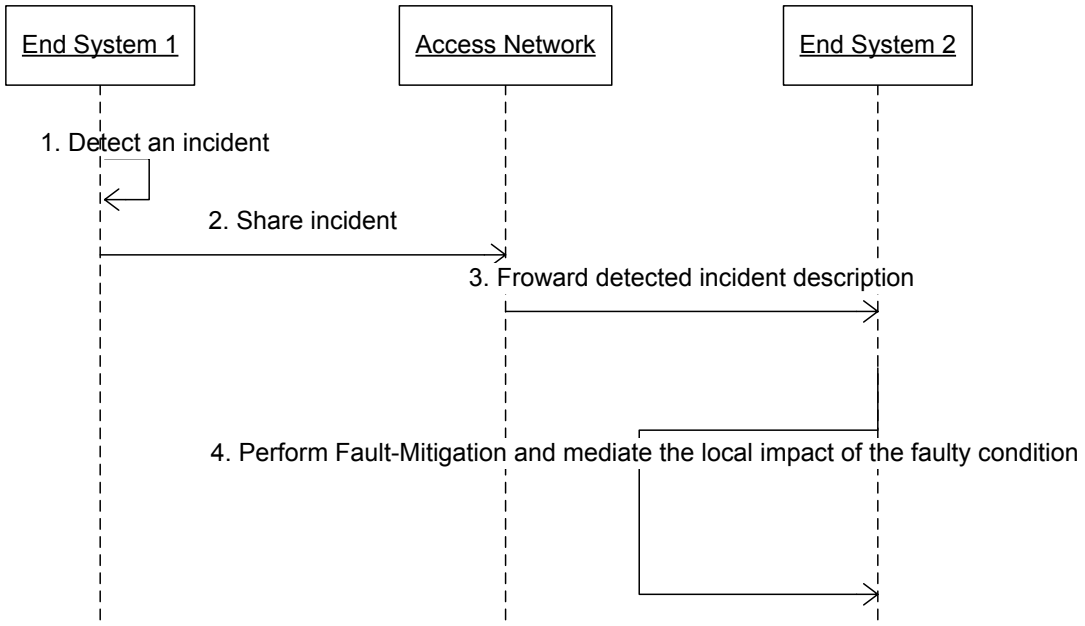

Figure F.8: Information Sharing between End Systems and Consequent Fault-Mitigation of the Local Impact of a Faulty Condition

based on FDH instances collaborating across multiple network segments.