INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2014; **27**:4103–4125
Published online 16 July 2013 in Wiley Online Library (wileyonlinelibrary.com). **DOI**: 10.1002/dac.2601

Framework for distributed autonomic self-healing in fixed IPv6 networks

Nikolay Tcholtchev*,† and Ina Schieferdecker

Fraunhofer FOKUS Institute for Open Communication Systems, Kaiserin-Augusta-Allee 31, 10589, Berlin, Germany

SUMMARY

A main requirement for the Future Internet is to enable self-management behaviors facilitating the network to adapt to changing conditions and self-heal in the face of erroneous states. On the basis of Autonomic Fault-Management principles, this paper proposes a framework consisting of a set of components operating inside the network elements and allowing the devices to collaboratively realize self-healing. In that context, Autonomic Fault-Management is intuitively constituted by the detection of the presence of faulty conditions, followed in turn by the self-diagnosis and identification of the corresponding root causes, and completed consequently by the removal of the identified root causes and their effects. The proposed framework implements a distributed control loop that interacts with the network operations personnel in case the current erroneous state is not resolvable by means of Autonomic Fault-Management. We argue that there are a number of mutual benefits between our proposed framework and the IPv6 protocol suite. This is demonstrated by a case study that illustrates these benefits and shows how the capabilities of IPv6 can be enhanced through the self-healing mechanisms of the proposed framework. Finally, the prototype implementation used for the case study is analyzed in terms of scalability and overhead produced in the network nodes. Copyright © 2013 John Wiley & Sons, Ltd.

Received 2 January 2013; Revised 4 June 2013; Accepted 11 June 2013

KEY WORDS: self-healing; framework; autonomic fault-management; distributed control loop; Fault-Isolation; Fault-Removal; Fault-Localization

1. INTRODUCTION

The ability of a network to employ self-healing mechanisms in order to preserve network functionality in the case of failure occurrences or malfunctioning is of great significance. Autonomic mechanisms applied to this task enable the network to self-manage with respect to erroneous states and at the same time reduce the involvement of the network operations personnel.

Autonomicity, as investigated in a number of recent research initiatives [1–4], is realized through the concept of a control loop. An autonomic control loop consists of different phases in which the state of the managed resources is observed and analyzed and, consequently, necessary actions are identified, planned and executed. Such control loop structures are also imminent for the field of Fault-Management, which is one of the most significant areas calling for further development in line with the concepts of Autonomic Networking/Computing. Because of its importance for today's and future networks, Fault-Management is also one of the main areas defined by ITU-T standard TMN (Telecommunication Management Network), and the FCAPS [5] network management framework.[a] On the basis of the FCAPS standard [5] and some key research activities in the area [5–7], the processes of Fault-Management can be synthetized as follows:

*Correspondence to: Nikolay Tcholtchev, Fraunhofer FOKUS Institute for Open Communication Systems, Kaiserin-Augusta-Allee 31, 10589, Berlin, Germany.
†E-mail: nikolay.tcholtchev@fokus.fraunhofer.de
[a]FCAPS stands for Fault-, Configuration-, Accounting-, Performance- and Security-Management.

Copyright © 2013 John Wiley & Sons, Ltd.

Fault-Detection – *detect the presence of a fault* (e.g. on the basis of some key performance indicators as handled in the area of Performance Management); Fault-Localization/Isolation/Diagnosis – *find the fault*; and Fault-Removal – *remove the fault*. Hence, an intuitive and naive notion of Autonomic Fault-Management (AFM) based self-healing is given by the idea of gluing together the aforementioned Fault-Detection, Fault-Isolation and Fault-Removal. In this work, we identify the processes and components, which are required to turn this intuitive AFM notion into reality; that is, the structures that enable AFM-based self-healing in a way that the resilience of a network is indeed increased for a variety of problems are investigated. In that context, we believe that AFM-based self-healing is better achieved in a distributed manner, within the network nodes, rather than going for an intrinsically centralized approach. The distributed aspect of the framework is vital as it aims at enabling the network nodes to collaborate and quickly resolve network problems thereby increasing the resilience of the network in question [8]. That is, the involved devices should be able to realize AFM-based self-healing by rapidly responding to failures without having the overhead of alarm aggregation, alarm suppression and dissemination to a central point as carried out traditionally in network management. By this, the network is expected to become more robust and provide an increased QoS, as it can quickly respond to challenging conditions and would escalate the erroneous state only in case the self-healing mechanisms fail to resolve it.

In this article, we conclude our research efforts, which were partially presented in previous conference and workshop publications [9–12], by specifying the 'nuts and bolts' of a framework that is capable of realizing self-healing based on distributed AFM. An initial draft [9] of our architectural framework is based on the Generic Autonomic Network Architecture [13] that was worked on in the scope of the EU-EFIPSANS [1] project. However, this version of our framework was specified without having gone through the process of evaluation and development of algorithms for the different components and without having the experiences from the case studies and the experiments presented in this paper. These experiences have led to the identification of new requirements and a redesign of the framework as presented in this article. Moreover, a Fault-Isolation algorithm [10] was specially developed for the goals of our framework. This algorithm is used as a library within the implementation of the prototype presented here and correspondingly plays a role for the reported numerical results. In addition, novel techniques [11] for the run-time synchronization of managerial actions in autonomic networks were developed for the purposes of AFM-based self-healing. The insights and the prototype from this work [11] were used and integrated as one of the components of the self-healing framework presented here. The interactions between the network operations personnel and our self-healing framework were also analyzed [12]. In the current paper, we do not discuss on that issue but rather refer to our previous publication [12] for more details. The added value of the current work lies in the details, the completeness and the final design decisions that were worked out after an implementation of the framework and experimentation in an IPv6 network. The various conducted experiments consist of a multifaceted case study being complemented by an experimental performance and overhead analysis. These new experimental results allow discussing and judging on the benefits brought by the proposed framework for the resilience and robustness of a communication network. In addition, we identify and motivate IPv6 as a suitable environment for realizing distributed AFM.

To summarize, the main goal of this work was to provide a sophisticated tool for increasing the resilience of IPv6-based communication networks and services by defining, implementing and evaluating a framework for self-healing that is based on distributed Autonomic Fault-Management principles.

The rest of this paper is organized as follows: Section 2 elaborates on the context and work related to the topic of AFM-based self-healing. Section 3 performs a requirements analysis for the architecture to be defined. Sections 4, 5 and 6 focus on defining the architectural framework and specifying the distributed control loop. Section 7 elaborates on the synergies between our framework and IPv6. The following sections deal with the prototype implementation and the evaluation of the proposed concepts. Finally, Section 12 concludes this article and points on future research directions.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

2. RESEARCH CONTEXT

2.1. From Fault-Management to autonomic self-healing

Self-healing naturally is concerned with handling faults, errors, failures and alarms. Therefore, we build on the definitions of those terms as consolidated and provided by Steinder and Sethi [7]: We understand a *fault* as the root cause for the erroneous functioning of a networked system. An *error* is a deviation in the conducted behavior of a component or a discrepancy with respect to the belonging-provided computation. In case of an error, the behavior or computation in question differs from the pre-specified functionality or expected values. An error either can directly originate from a fault or can be a consequence from another error. Finally, an error propagates further as other error(s) until resulting in a failure. A *failure* is regarded as a 'visible' service error; that is, it is a deviation between the expected and pre-specified result of a service and the delivered one. In the scope of this paper, faults/errors/failures are often referred to as *incidents*. In addition, *alarms* are understood as notifications indicating abnormal conditions. An alarm may or may not stand for an incident (fault/error/failure). For example, an alarm notification would be issued if a particular threshold is being approached, which in this case does not mean that a fault has occurred.

Taking AFM as a building block for self-healing and coming back to the fact that Fault-Management is traditionally considered as consisting of *Fault-Detection*, *Fault-Isolation/Localization/Diagnosis*, and *Fault-Removal*, we can now characterize it by explaining each of its processes on the basis of the aforementioned view on faults/errors/failures/alarms. *Fault-Detection* stands for the process of detecting that a fault has happened. That is, a failure can be detected by a functional entity. We emphasize here that the process of Fault-Detection is the one that gives the indication that a fault has occurred in the network. The process of *Fault-Isolation* deals with analyzing the amount of detected incidents and alarms and identifying the root cause (i.e. faults) for the experienced erroneous state. On the basis of the Fault-Isolation results, a strategy to correct the faults (root causes) and handle the belonging errors and failures is implemented such that the system can return to normal operation. In current practices, the aforementioned Fault-Management processes are mainly manually conducted, with the aid of different management tools. Going a step beyond, the combination and interplay of Fault-Detection, Fault-Isolation and Fault-Removal without the need for human intervention is what we denote as AFM-based self-healing. The realization of self-healing based on distributed AFM is a complex task because it requires the collaboration between different network devices, and the distributed realization of the aforementioned interplay of Fault-Management processes.

2.2. Related work

The main research in the area of Fault-Management has concentrated on different techniques for Fault-Detection [14, 15] and Fault-Isolation [7, 16]. These two processes were intensively researched during the past years. Some samples of impressive research works: Generic frameworks [6, 17] for Alarm Correlation were worked out in the early years of automating network management tasks. Bayesian Networks were instrumented for the purpose of Fault-Isolation in 3G networks [18]. Graph theory approaches such as bipartite graphs were used for efficient network black hole[b] Fault-Localization [14]. Advanced Frameworks for efficient Fault-Diagnosis were designed, for example, on the basis of a mixture of passive probabilistic approaches and active probing [16]. The efficient monitoring and alarm correlation in End-to-end Ethernet networks was investigated, and an architectural framework for these purposes was defined [19]. The designed architecture [19] facilitates the network administrator to take an adequate decision in order to remediate the challenges faced by the network. In addition, the work of Steinder and Sethi provides a thorough survey [7] of methods for Fault-Detection followed by an Event Correlation process in telecommunication networks.

[b]The term 'black hole' stands for a network component that drops packets without giving proper notifications of packet loss.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

A number of research activities related to AFM have been ongoing during the recent years. In the work of Sterritt [20], a Unified Fault-Management framework for the telecom domain is extended by introducing the notion of reflex reactions (inspired by the analogous processes in the human body) triggered by the results of a Heart-Beat monitoring component. This is a first simplified step toward developing a theory around the notion of AFM. Solid research [21, 22] has been conducted toward the implementation of distributed Fault-Management based on intelligent software agents. This work mainly focuses on the efficient decentralized Fault-Isolation, that is, as close as possible to the devices, and derives a probabilistic framework that uses the concept of entropy, in order to determine the quality of root cause analysis results. The application of AFM to Wireless Mesh Networks was also a topic of interest in the recent years [23]. The authors of this research [23] define a set of processes and methods, which in their view, should constitute or be applied to AFM in Wireless Mesh Networks, namely, Network Measurements for Fault-Detection, diverse Fault-Diagnosis Algorithms, and finally Autonomic Fault-Recovery. A number of challenges regarding the processes and methods are identified; for example, (i) the Fault-Diagnosis process is expected to be extremely time and resource consuming due to the amount of data to digest and the complexity of the reasoning algorithms, and (ii) the danger of executing interfering actions in the process of Autonomic Fault-Recovery in a way that AFM is even harmful to the network. However, this paper does not prescribe an architecture that has the capability to implement AFM and does not provide any examples of use cases or even experimental case studies. Furthermore, the UniFAFF [24, 25] constitutes a framework for the realization of AFM and Failure Detection in clean slate networks, where functional entities can be designed in a way as to collaborate toward the resolution of faulty conditions. All the presented approaches hitherto are successful initial analytical studies of the application of AFM to the domain of specific networking technologies. In our research, inspired by the aforementioned works, we specify in detail the elements of a distributed control loop for AFM-based self-healing in fixed IPv6 networks and refine it on the basis of a prototype and a set of experiments, which have shown to us where additional components and algorithms are required. This allows for implementing efficient self-healing procedures in the network nodes.

3. REQUIREMENTS ANALYSIS

In this section, a list of requirements that determines various aspects, components and mechanisms of the emerging AFH (*A*FM-based *F*ramework for Self-*H*ealing) is derived. We hold that this is quite an exhaustive list reflecting the plethora of issues related to reactive self-healing, on the basis of our literature review, discussions in the scope of projects such as ANA [26] and EFIPSANS [1], and most importantly, on the basis of experiences gained in experimenting with early and mature prototypes of the framework described in this article.

First of all, the AFH must provide interfaces and components addressing the tasks of Fault/Incident-Detection, Incident/Alarm-Dissemination across the network, Fault-Isolation, and Fault-Removal [23, 27, 28]. For the purpose of Fault-Isolation, the causality relations (Fault-Propagation Models [FPM]) 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 [7]. In addition, the results of the Fault-Isolation process within a network node (e.g. router) should be verified locally, such that resulting Fault-Removal actions are avoided in case of incorrect Fault-Isolation. In the course of experimenting with the early prototypes of the framework, we had the chance to experience such situations and had improved the design of our framework accordingly. In the context of Fault-Removal, the framework must take into account the dynamics of the node and the network with respect to the history of executed management actions. For example, while experimenting with our prototype, we experienced that in some cases, if a Network Interface Card (NIC) has been restarted a number of times, the driver is not able to reinitialize, and hence, only the rebooting of a node can be used as a last resort to restoring full functionality. In addition, it is required to ensure the stability with respect to parallel/concurrently running AFM processes (e.g. in a multi-threading environment). Hence, the Fault-Removal module inside the architecture needs to guarantee that

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

the overall set of actions to be executed is synchronized toward a common goal of improving the overall fitness of the network [23]. Our experiences with early prototypes of the framework have shown that this is extremely important in a distributed environment where events can be delayed for different reasons leading to contradicting and undesirable combinations of corrective tentative actions. Hence, a component that synchronizes, in an appropriate way, the actions issued by diverse parallel/concurrent self-healing processes is required. In addition, after executing an action on a single 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 AFH aware of the success of its activities and will contribute to the stability of the framework by providing means to assess the quality and the trends in the undertaken autonomic managerial actions. Furthermore, our experiments have shown that there will be cases in which the AFH will not be successful in correctly realizing self-healing, for example because of any circumstances, which are not captured in the employed FPM, or because of incomplete and/or inaccurate symptoms. Therefore, the framework should be able to determine if a particular situation is beyond its capabilities and should escalate it to the network operations personnel.

In the following, a list of tangible requirements, drawn from the previous discussion, is specified:

Req.1 The proposed architecture must define components and interfaces that target the processes of Fault-Detection, Incident/Alarm-Dissemination across the network, Fault-Isolation, and Fault-Removal.
Req.2 The proposed framework should remove problems operating as close as possible to the device, thereby working in a distributed manner without (fully) relying on a centralized instance for realizing AFM processes.
Req.3 The architecture must provide components and interfaces to enable the collaborative sharing of alarm/incident information inside a device and across the network.
Req.4 The framework must consider the causality relations (FPM [7]) regarding root causes (faults) and their propagation as one or many errors until eventually resulting in observable failures and alarms.
Req.5 The results of the Fault-Isolation process should be verified in order to avoid Fault-Removal strategies on the basis of incorrect suggestions regarding isolated faults. Such Fault-Removal actions might be even damaging to the network integrity. In critical cases of incorrect Fault-Isolation results, the situation should be escalated to the network operations personnel. The verification strategies are to be supplied by the network operator, and thus, the framework should provide the possibility for such configurations [12].
Req.6 The architecture needs to ensure that the actions undertaken by parallel/concurrent self-healing tprocesses do not contradict each other and that the entire decision-making process effectively contributes to the overall fitness of the network.
Req.7 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.
For example, in some cases, if a 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.
Req.8 The effectiveness of the undertaken management actions must be assessed.
Req.9 The key parts of the distributed AFM control loop should be realized through scalable and efficient algorithms thereby reducing the overhead, for example in terms of memory consumption, within a network node.
Req.10 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 operations personnel.

The 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.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

4. DISTRIBUTED CONTROL LOOP DESIGN: A HIGH LEVEL VIEW

Within this section, the distributed control loop (satisfying Req.2) is drafted without going into the details of the involved components. That way, we want to give the reader a high level idea of our approach before going into the 'nuts and bolts' of AFH. We presume that each AFH node is equipped with an entity (AFM agent[c]) that can potentially realize the processes of Fault-Isolation and Fault-Removal on the basis of the corresponding models, for example FPM or Fault-Removal policy models. Another required assumption is that some of the nodes have 'monitoring sensors' deployed, that is, monitoring components that are able to detect particular incidents and alarms indicating the presence of faulty conditions. Such detected alarms/incidents are described in a specific format and pushed first to the local AFM agent. Certain faulty conditions may concern only the node on which they were detected. In such case, the AFM agent of the node proceeds locally with the process of Fault-Isolation[d] and eventually Fault-Removal. In case the detected incidents are related to a network wide erroneous state, they are conveyed to another local agent that takes care of disseminating the alarm/incident descriptions within the network scope in question, for example a Local Area Network. That is, all the relevant nodes within the scope in question are aiming to converge with respect to their view on the set of corresponding incidents and alarms. This gives them the appropriate knowledge as to proceed locally with the process of Fault-Isolation. As a result, the AFM agents within the nodes periodically trigger a Fault-Isolation task based on the set of alarm/incident events, which have been either detected locally or have been received from the network. The Fault-Isolation processes in all nodes run on the basis of the same FPM, a copy of which is stored locally within each node. This Fault-Propagation model is provided by the network manager on the basis of 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 incidents and alarms (which might not always be the case), every AFM agent in every node would arrive at the same conclusion regarding the root cause of the erroneous state, because of the Fault-Isolation process based on the same FPM deployed in each device. The results of this root cause analysis are then forwarded to the Fault-Removal Functions (FRF) of the AFM agent. This FRF module operates on the basis of 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 because 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 remove the identified fault(s), the situation has to be escalated and the network manager notified [12]. In case of successful Fault-Removal, a recovery notification is sent across the network scope in question. The described steps hitherto indicate that the distributed control loop consists of local control loops, which are executed on the basis of the alarm/incident information detected on a node or shared between nodes.

5. COMPONENTS SPECIFICATION ON NODE LEVEL

We first focus on the architectural aspects of AFH in terms of components required within a node in order to realize AFM-based self-healing. We base our descriptions on the AFH node architecture presented in Figure 1. Thereby, we investigate in turn the components, which realize the different subtasks of the distributed control loop presented in the previous section. Figure 1 classifies these components and illustrates the modules responsible for the storage and dissemination of events – alarms and incidents – in dotted boxes,[e] whereas the components that consume the provided

[c]By *agent*, we denote a computer program (component) that acts autonomously (i.e. perceiving the environment and acting), irrespective from whether it has any inbuilt 'intelligence' or not. Refer to (Stuart Russell & Peter Norvig, "Artificial Intelligence: A Modern Approach", (3rd Edition), December 11, 2009, Publisher: Prentice Hall, ISBN-13: 978-0136042594) for a detailed classification of different types of agents.

[d]Within the Fault-Propagation Model, such incidents would be modeled as not having any type of relation to incidents from other network nodes.

[e]All the components realizing the distributed control loop, that is, those in solid line boxes, are the contribution of our research. The concepts around the storage and dissemination components, that is, those in the dotted boxes, are borrowed from state-of-the-art. An exception here is the Garbage Collector that emerged as a requirement after experiments with early AFH prototypes.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

alarm/incident information and realize the distributed autonomic control loop are presented in solid line boxes. We start in the sequence of describing the distributed control loop thereby focusing in turn on the components responsible for the detection of a faulty condition, the storage and dissemination of the belonging alarm/incident descriptions, and proceed with those architectural parts responsible for Fault-Isolation, Fault-Isolation Assessment, Fault-Removal, Action Synchronization (i.e. control loop stability) and Fault-Removal Assessment.

5.1. Fault-Detection Agent

The entity responsible for coordinating the Fault-Detection efforts inside a network node is the Fault-Detection Agent (FDA) illustrated in Figure 1. The FDA enables the processes of Fault-Detection and collaborative sharing of incident information and links to Req.1 and Req.3 from Section 3. The role of the FDA is to satisfy the monitoring requests and needs for tackling the erroneous states resolvable by means of AFH. That is, this entity starts monitoring jobs, which observe the status of different Key Performance Indicators (KPIs), and can recognize anomalous states indicating the presence of faults within the system. Such KPIs might be for example the counters (e.g. dropped packets or frame collisions) of a NIC, different traffic metrics for example duplicate acknowledgments or retransmissions in TCP flows, as well as traditional QoS metrics as known within telecommunications, that is, jitter, delay, out of order packets, and packet loss.

The FDA orchestrates and regulates the behavior of different monitoring tools as to instrument passive and active monitoring techniques, in order to gain the required information. In addition, there is also a direct communication channel between the FDA and a set of lightweight alarm and incident repositories, which participate actively in the alarm and incident information sharing within a node and across the network scope in question. That is, once an alarm/incident has been detected, it is described on the basis of an information model and stored in the corresponding repository by the detecting monitoring entity, which is orchestrated by the FDA. Thus, the next section describes the role of the alarm and incident repositories within a node.

5.2. Alarm and incident repositories

The alarm/incident repositories (in the center of Figure 1) inside the device architecture are the consolidating point for faults/errors/failures/alarms coming from the node and from the network scope. In that sense, this set of components meets aspects of Req.1 and Req.3 from Section 3.

The alarm/incident 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. Such a structure

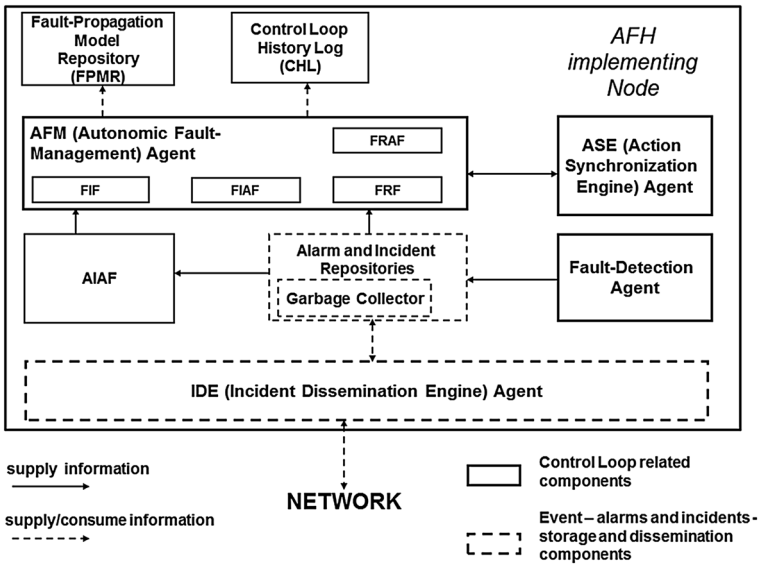

Figure 1. Node components of the AFH Framework.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

was proposed in previous research efforts [24], and on the basis of it, we adopt the following repositories for the purpose of AFH: (i) *Local and External Alarms Repository*; (ii) *Local Incidents Repository*; (iii) *External Incidents Repository* – dedicated to storing incident information related to other network nodes; and (iv) *Asserted Incidents Repository* – responsible for storing incident information coming from other network nodes and related to the local node.

The alarm/incident information being reported to the repositories may trigger local probing mechanisms 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, which is a separate agent that triggers different probing mechanisms based on a reaction model supplied by the network operations personnel.

The repositories should keep a list of other nodes that might be interested in receiving certain types of locally detected alarms/incidents. On the basis of this list, the repositories should actively trigger alarm/incident dissemination across the network, over the Incident Dissemination Engine (IDE) (described within the next sections), once a corresponding locally detected alarm/incident has been reported. The repositories are also responsible for forwarding submitted alarm and incident descriptions to the AFM agent described in the next section. To summarize, once an alarm or incident has been (i) locally detected by the corresponding monitoring entity (orchestrated by the FDA) and (ii) submitted to the repository in question, (iii) the belonging description is automatically forwarded to the IDE, such that it is disseminated across the network – in case the alarm/incident information is of interest for other network nodes, and (iv) to the local AFM agent to trigger local self-healing processes.

In case a fault has been removed and a recovery notification has been issued, the repository should schedule the related alarms and incidents for deletion by the Garbage Collector.

5.3. Autonomic Fault-Management agent

The AFM agent within a node is responsible for the autonomic interplay of Fault-Detection, Fault-Isolation and Fault-Removal with respect to faulty conditions, which affect the local device. As indicated in figure 1, these processes are assigned (1) to the Fault-Isolation Functions (FIF) – which is the component gluing together Fault-Detection and Fault-Isolation, and (2) to the FRF – for removing faults and their effects locally. In addition, as Fault-Management turns autonomic, the tasks 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), all of them being part of the AFM agent as indicated in Figure 1. Next, all these components are described in turn.

5.3.1. Fault-Isolation Functions. The FIF module of the AFM agent is the component responsible for isolating and diagnosing faults in the process chain of AFM-based self-healing. The FIF is the subcomponent that fulfills Req.4 from Section 3, which is related to causality relations between symptoms and root causes. In addition, the FIF fulfills aspects of Req.1, especially the demand for components for Fault-Isolation.

The FIF functions are realized by a library, which is invoked by the AFM agent after some alarms/incidents have been reported and their analysis is correspondingly required. The alarm/incident description can be either referring to locally detected events, which are forwarded to the AFM agent by the corresponding repository, or to alarms/incidents, which were detected elsewhere in the network and were reported locally using the service of the IDE (see Section 5.7). The FIF functions gather such alarm/incident descriptions and trigger the Fault-Isolation process according to some predefined conditions: (i) Fault-Isolation is triggered *every time a particular predefined number of incidents or alarms have been reported*; (ii) Fault-Isolation may also be immediately initiated after a single alarm/incident event has been reported given that the event has a *high (according to a predefined value) severity level*; and (iii) if only few (less than the predefined number of) alarms/incidents are reported and their severity is not causing the FIF functions to immediately trigger Fault-Isolation, then Fault-Diagnosis is triggered *after a predefined time slice, which can be realized by the expiration of a timer*. The aforementioned number of alarms/incidents, the severity level of the alarms/incidents immediately triggering Fault-Isolation, and the duration length of

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

the time slice, after which the FIF functions initiate Fault-Isolation, are parameters, which need to be supplied by the human experts tweaking the AFM-based self-healing. Once Fault-Diagnosis is triggered, selected algorithms start reasoning on the basis of the knowledge 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.

5.3.2. 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 before issuing a Fault-Removal action. These uncertainties arise because of the stochastic nature and complexity of the network and deployed protocols. 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 FIAF is the component responsible for avoiding such situations thereby addressing Req.5 from the list of requirements. In the course of its operation, the FIAF can for example deploy different probing mechanisms, check the running configuration of some entities, and examine the queue sizes or the state of a NIC, in order to test and verify the results of the Fault-Isolation task. Given that the Fault-Isolation result is wrong, the situation should be escalated to the network operations personnel, in order to indicate that the employed FPM is insufficient. Further criteria for escalating faulty conditions to the network administrator are specified in our previous research work [12].

5.3.3. Fault-Removal Functions. Given the verified results of the Fault-Isolation process, the FRF of the AFM agent 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 of the self-healing control loop. In order to ensure stability with regard to avoiding contradicting actions issued by parallel (e.g. on a multi-processor device) or concurrent (e.g. in a multi-threaded environment) instances of the AFM control loop, the FRF refers to the Action Synchronization Engine (ASE). The ASE is responsible for synchronizing such tentative actions and for selecting an optimal subset (from the tentative actions) on the basis of the goal of improving the performance of the network. The ASE component is described in the coming sections. Given that an action has been allowed by the ASE, the FRF triggers the execution of the corresponding scripts or management tools 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 [12] which is supplied by the human experts tweaking the AFM-based self-healing of the network.

5.3.4. Fault-Removal Assessment Functions. The FRAF functions have the task to inspect (according to Req.8 from Section 3) whether the Fault-Removal process was successful or not. 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. On the basis of the result from this assessment, a local FRAF escalates the problem to the network operations personnel in case Fault-Removal has failed [12]. Alternatively, in case Fault-Removal was successful, the FRAF marks as cleared/recovered all the incidents and alarms related to the problem in question and triggers the dissemination of a recovery notification over the IDE. Finally, the control loop's execution, with its status obtained by the FRAF, is logged to the local *Control Loop History Log* (CHL) described in the coming sections.

5.4. Action Synchronization Engine

The ASE is the component responsible for ensuring the stability of parallel or concurrent local instances of the AFM control loop thereby meeting the needs of Req.6, which demands that the tentative Fault-Removal actions of parallel/concurrent AFM processes should be synchronized toward a common goal. The task of the ASE is to 'gather' tentative actions and to synchronize them toward a common goal on the basis of a predefined utility function and optimization problem.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

Similar with the case of the FIF, the actions are gathered over a *predefined period* after which the synchronization is initiated. A synchronization of tentative actions can be also triggered after a *predefined number of synchronization requests* have been submitted, or in the case when a synchronization request is marked as urgent. The ASE allows only those tentative actions to be executed, which do not interfere with each other and contribute to the optimization of the corresponding utility function, that is, maximization of the underlying network fitness model. Besides the synchronization of local node actions, a centralized ASE might be put in place as to synchronize tentative actions to be carried out in different network nodes. Details on the structures and the algorithm applied to that issue were presented in our previous research work [11].

5.5. Fault-Propagation Model Repository

The FPMR is the AFH node component responsible for storing a local node instance of the FPM for the network scope in question. The FPM stores the causality relations between different failures and alarms, intermediate events in a fault-propagation, that is, errors, and the corresponding root causes (faults). These relations are stored in a way that allows for fast real-time reasoning and root-cause analysis. Hence, the FIF functions of the AFM agent 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.

5.6. 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 CHL and can serve different purposes. On one hand, they can be used by the network operations personnel to understand and improve the operation of AFH within the network. For example, the FPM may be improved as to deliver better Fault-Isolation results, or the different policy models specifying the actions at different stages of the control loop (Fault-Removal, Fault-Isolation Assessment, etc.) can be further refined. The control loop history should also be easily accessible over an API as to give the FRF functions the capability to be aware of the type of actions, which were already executed by the AFH. On the basis of the interplay between FRF and CHL, AFH is supported in addressing Req.7 from Section 3, which demands that the dynamics of the managed resources should be considered. The experiences, drawn from experiments with our AFH prototype, have shown that the interplay between CHL and FRF 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 to be restarted in case a following Fault-Removal action is again to be executed on it.

5.7. Incident Information Dissemination Engine

The Incident Information Dissemination Engine (IDE) is the entity that plays the role of a gateway to the network inside an AFH implementing device. It is responsible for disseminating and receiving alarm/incident information and recovery (i.e. Fault-Removal) notifications to and from the network and meets the requirements for Incident/Alarm-Dissemination (part of Req.1) as well as for facilitating the collaborative sharing of alarm/incident information according to Req.3 from Section 3. Thus, once an alarm/incident manifesting a faulty condition has been observed, described, and submitted to the alarm/incident repositories on the node hosting the corresponding Fault-Detection component, the repository in question forwards the alarm/incident description to the AFM agent and to the local IDE. The IDE in turn disseminates the description to involved nodes across the network scope where Fault-Isolation is to be triggered on the basis of it. On the other hand, when information is received from the network, the IDE has the responsibility of storing it in the appropriate local node alarm/incident repository and conveying it to the local AFM agent such that this alarm/incident description can be considered within the next Fault-Isolation. Finally, after a successful completion and assessment of a

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

self-healing process, a recovery notification is sent (over the IDE) across the network scope such that the relating alarm/incident descriptions can be deleted across the repositories of the involved nodes.

6. DYNAMIC ASPECTS OF THE FRAMEWORK

On the basis of the architectural descriptions provided in the previous section, we proceed with outlining the dynamic aspects of the AFH framework and specifying the distributed control loop in detail. Because of the limited space, we focus on one particular interaction flow given as a sequence diagram in Figure 2. This sequence describes the interplay between the AFH components within a device in case an incident was detected on a remote node in the network and was reported to the local node over the services of the IDE compartment. The components in Figure 2 are presented with the abbreviations that were established in the previous section. Furthermore, we summarize the Incident Repositories as NAIR, which stands for *Node local Alarm and Incident Repositories*. In addition, we use two more objects, one of which refers to the Network (NET) and is used to indicate that a message has arrived at the node from the network, and a second one that stands for the local environment (LE), that is, the set of node local functional entities.

Given that an incident has been detected somewhere in the network by an entity orchestrated by the belonging FDA, this incident is consequently disseminated across the network scope by using the services of the IDE compartment. Once the incident message arrives at the local node (*reportIncident* in Figure 2.), it is accepted by the local IDE and forwarded (*storeIncidentInformation* in Figure 2) to the NAIR. The NAIR in turn informs the local AFM agent (*informAFMagent*), which would trigger Fault-Isolation according to the conditions specified in the previous section. In this case, that would either mean that a pre-specified number of incident descriptions are already pending and are waiting to be analyzed, or that the submitted incident is deemed as urgent by the detecting component on the remote node. If none of these conditions is met, the incident is considered for Fault-Isolation by

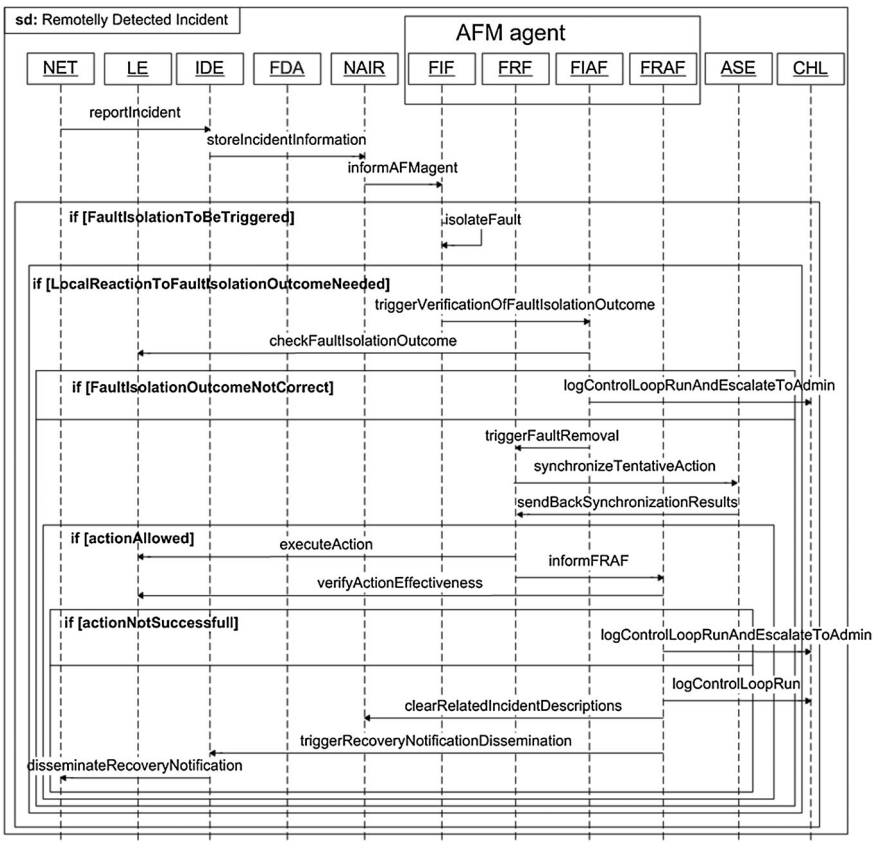

Figure 2. Sequence diagram specifying the processing of a remotely detected incident.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

the FIF of the AFM agent as soon as Fault-Diagnosis is triggered by the expiration of a timer (see previous section). After, Fault-Isolation has been executed and the outcome is available, its correctness is checked (*checkFaultIsolationOutcome*) by the FIAF in case that an action is to be executed locally as a result of the Fault-Isolation outcome. In case the Fault-Isolation results are not correct, the control loop execution is recorded to the log (CHL), and the faulty condition escalated to the network operations personnel (Req.10 from Section 3). In case of correct Fault-Isolation results, the corresponding local reaction is triggered. Before the belonging action is performed, it needs to be allowed by the local ASE agent. If the action is allowed, then it is executed (*executeAction*) by FRF, and subsequently, the FRAF check (*verifyActionEffectiveness*) whether it has been successfully executed, and the faults were indeed removed. Given that the Fault-Removal action has failed, the control loop execution is recorded in CHL and escalated to the network administrator (Req.10 from Section 3). In case of a successful Fault-Removal action, the control loop execution is logged to the CHL (*logControlLoopRun*), and all incident descriptions related to this fault are cleared from the local repositories (*clearRelatedIncidentDescriptions*). Subsequently, a recovery notification is sent across the network (*triggerRecoveryNotificationDissemination* and *disseminateRecoveryNotification*) such that all involved nodes can also clear the related incidents from their local NAIR, on the basis of the functionality of the local Garbage Collector. These interaction flows are based on our initial considerations and on the lessons learnt from our prototype implementation and corresponding trials.

7. THE ROLE OF IPV6 IN ENABLING AUTONOMIC SELF-HEALING

The specification of AFH presented so far allows us to consider the suitability of different network environments for realizing the processes of AFM-based self-healing. Communication networks are clearly moving toward all IP network infrastructures, which allows the implementation of NGN and multiservice (i.e. voice, data, video, etc.) Future Internet type of networks. In that context, IP is used as the glue between the transport layer and the underlying network technologies, such as Ethernet, SDH, and Frame Relay. Clearly, IPv6 constitutes the set of latest advances in IP research and development, which have also successfully passed the process of standardization at IETF. Moreover, IPv6 is being intensively researched, for example, analysis of its QoS, performance and interoperability characteristics [29, 30], and novel mechanisms are built on top of it, such as improved address auto configuration for mobile ad hoc networks [31]. Moreover, intensive research activities [32–34] are being conducted toward establishing IPv6 as a key protocol for wireless sensor networks. Further, we recognize a number of features within IPv6, which increase the effectiveness and quality of AFM as compared with a traditional IPv4 network. The benefits brought by IPv6 features in the realization of AFM-based self-healing are the following: (1) The IPv6 Multicast-Listener-Discovery protocol [RFC2710] facilitates a multicast structure for scalable and efficient mechanism for dissemination of alarm/incident information. (2) The true end-to-end communication paradigm provided by IPv6 gives the possibility to directly identify sources of flows of interest to the AFM machinery of the whole network. Assuming that even the end systems are equipped with AFH components, then AFH components inside the network (e.g. in routers) are able to influence the AFH components in end systems identified as transmitting flows that may be affected by incidents inside the network (e.g. in the core). (3) The concept of 'Scope' [RFC4291] that comes with the IPv6 Addressing Architecture is very useful in the context of the dissemination of alarm/incident information to some scope, for example *link-local-scope*. (4) The IPv6 Hop-By-Hop Option [RFC2460] can be exploited for the purpose of collaborative information sharing across every hop along a path [35]. (5) The presence of inbuilt security mechanisms in IPv6, that is, IPsec [RFC1825] [RFC1829], allows for secure alarm/incident information sharing among the involved nodes thereby facilitating the confidentiality and integrity regarding the alarm/incident descriptions in the network. (6) The Secure Neighbor Discovery protocol of the IPv6 suite minimizes the probability for neighbor spoofing attacks (e.g. ARP spoofing in IPv4) and contributes to the secure exchange of alarm/incident information thereby preventing attacks on the AFH control loop structure. Features, (2),(3),(5), and (6) are not present or only limited in IPv4. In addition, it can be observed that IPv6 provides better means for alarm/incident information sharing among AFH nodes – (1),(2),(3), and (4), thereby influencing positively the implementation of related

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

requirements (Req.1, Req.3, and Req.9) on AFH. Hence, we observe that IPv6 is superior to IPv4 when it comes to enabling AFM-based self-healing. In addition, the way IPv6 benefits from AFH is exemplified by the case study presented later on.

8. IMPLEMENTATION AND TESTBED ENVIRONMENT

In order to validate our approach, we implemented the components of AFH by using C\C++ and Java on a Linux platform such that we can easily deploy them on the Linux based routers inside the testbed illustrated in Figure 3. The orchestration part of the FDA was developed in Java and executed (interpreted) in the Java Virtual Machine, since we considered it as not so time-critical given the ultimate goal to implement the control loop in a way that it operates as fast as possible. This is backed by the assumption that the orchestration part is only going to start some *monitoring sensors* at the bootstrap phase of AFH inside a node and then let them do the job and directly report detected alarms/incidents to the local repositories. Besides this orchestration part of the FDA, all other entities are implemented in C\C++. For the monitoring components, we wrap different freely available monitoring tools (e.g. *ethtool*, *tcpdump*, *tcptrace*) in small C programs and automate the interplay among the tools as to extract the required information, which potentially contains indications for faulty conditions. The agents and the different repositories are all realized as daemons running on top of the operating system. The communication between the FPM repository and the AFM agent is realized using shared memory, as this allows the AFM agent to directly access the FPM every time Fault-Isolation is required. For the communication between the rest of the functional entities within the node architecture, Unix Domain Sockets were utilized, which enabled fast message exchange inside a node. For the communication among the IDEs, residing in different nodes, we use UDP as a protocol for communicating short datagrams containing small in size alarm/incident descriptions. The usage of connection-oriented communication (e.g. TCP or the reliable Pragmatic Generic Multicast – PGM [RFC3208]) is a challenge in that context because alarms/incidents can occur sporadically and the maintenance of a data stream is not so trivial, for example in PGM, the delivery of a non-ACK would not work reliably given the

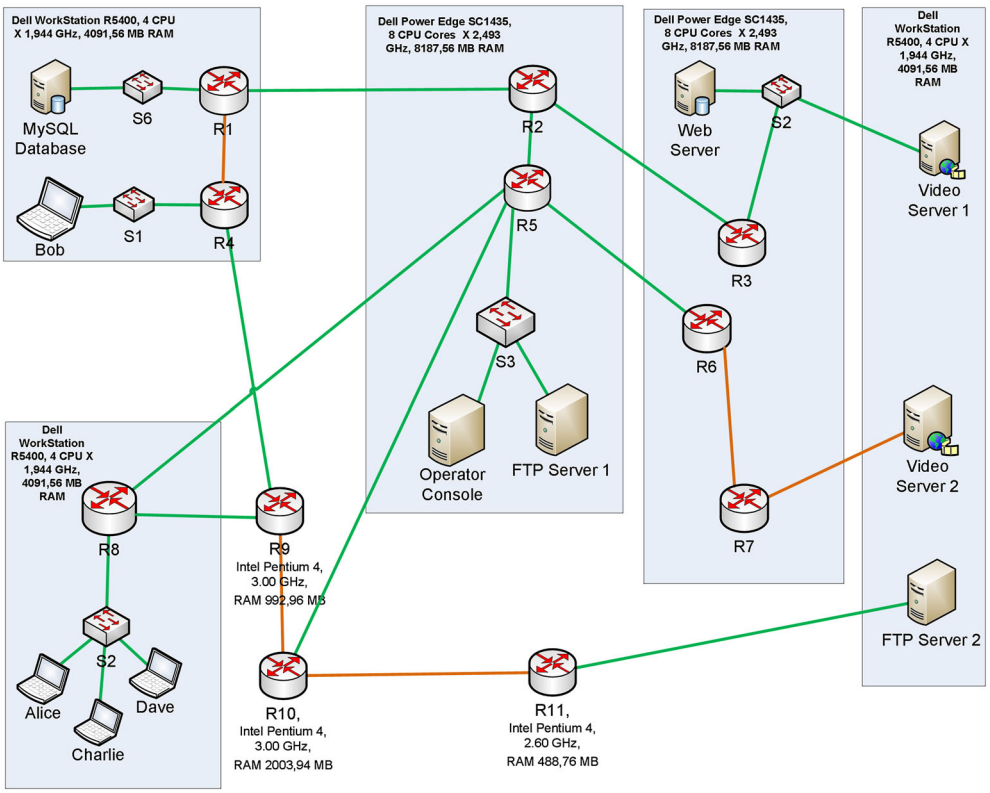

Figure 3. The testbed for experimenting with AFH.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

sporadic nature of the alarm/incident data to send. In order to increase the chance of complete successful alarm/incident dissemination, we implemented a type of canonical flooding where the node that has detected the alarm/incident sends the resulting short message (using its IDE component) to a list of addresses (potentially including multicast addresses). In turn, every recipient node forwards again the message to all the addresses, in case the alarm/incident description contained in the message is unknown to it, or drops it in case it has already received it. We simulated this procedure by using the OMNET++ [36] simulator and the resulting scalability analysis is provided in Section 10. Regarding the ASE agent, we implemented the procedure for the synchronization of tentative actions, which was proposed in our previous work [11]. This procedure boils down to solving a binary optimization problem, for which we employed the Coin-OR solver [37]. The Fault-Isolation within the AFM agent was realized by using the outcome of our previous research [10], where we defined a reasoning scheme on the basis of Markov Chain principles. Thereby, we discovered that our algorithm shows better scalability than traditional Bayesian Networks [10]. For the FRF, FRAF, FIAF, and ASAF 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 [12]. The alarm/incident repositories were implemented as lightweight daemons for storing the alarm/incident description in a hash type of a data structure. The architecture was also integrated with a freely available Network Management System – *Nagios*. In order to achieve this, some Nagios plugins that communicate with the components across the network and realize the escalation of faulty conditions, which are not resolvable by the AFH mechanisms within the nodes, were developed.

Finally, some words regarding the testbed (Figure 3) on which we tested and experimented with the AFM-based self-healing framework. This testbed was developed at the Fraunhofer FOKUS institute in the course of the EFIPSANS project. It consists of virtual machines (those in gray boxes) being hosted within a *Vmware ESX* server environment. In addition, a number of physical machines were integrated (those out of the gray boxes), in order to explore different situations on the basis of real (and not only emulated) hardware. The visualized network in Figure 3 is an IPv6 network, whereas in parallel, a management network is running, which allows us to access the machines, deploy our prototype, and run experiments on the IPv6 infrastructure. The nodes have Linux Ubuntu installed, and the routers are running the Quagga routing suite whereby the Open Shortest Path First version 3 (OSPFv3 – the IPv6 ready OSPF) routing protocol is executed. The green links in Figure 3 denote Gigabit Ethernet (GE) links, whereas the rest are all Fast Ethernet (FE) links. The different type of links play a vital role during the experiments, as the difference in the offered MTUs of GE and FE links – GE links offer a higher MTU than FE links – creates a potential for IP Black Holes as described in the next section.

9. COMBINED CASE STUDY IN FIXED IPV6 NETWORKS

The AFH framework was applied to a combined case study, involving a combination of different networking problems being introduced to the experimental IPv6 network in Figure 3. The problems were selected on the basis of a survey conducted among the partners in the scope of the EU-EFIPSANS project [1]. For the sake of the case study, the networking problems and corresponding faults were injected by an automated script that was issuing commands on the network nodes in Figure 3 over the management interfaces. Four different types of problems were introduced at multiple points in the network including (i) Black Holes [RFC2923] in IPv6 on *R9* and *R1*, (ii) Ethernet Duplex Mismatches [38] on links *R9*↔*R10* and *R10*↔*R11*, (iii) large packet delays resulting in low quality of the link *R1*↔*R2*, as well as a (iv) crash of the Domain Name System (DNS) server running in sub network S3. This results in an overall number of six network faults being injected by an automated script at different points (links, routers, and servers) in the network in Figure 3. Correspondingly, an FPM comprising around 40 events was constructed as to facilitate the Fault-Isolation process in the network nodes. Next, we elucidate on each of the network problems in turn.

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 be lost causing the network to fail delivering its services. [RFC2923] describes a specific type of a Black Hole phenomenon that can potentially occur in IPv6 networks, which have inherited their firewall

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

configurations from previous IPv4 deployment, for example because of 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 Path Maximum Transmission Unit changes during the lifetime of a flow, such that the sender can adjust the size of the packets it sends out. This problem can lead to extensive packet loss in 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 because 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 [RFC4942]. In our case study, we instrumented monitoring sensors that were monitoring TCP flows on R1 and R4 in order to detect *duplicate acks* or *sudden drops in TCP traffic* that are indications for Black Hole problems. The FPM was used to correlate multiple incidents and to isolate the Black Hole and correspondingly reconfigure the firewall in question (on *R9* or *R1*) according to corresponding recommendations [RFC4942].

Duplex Mismatches can occur in IEEE 802.3 Ethernet environments, where the NICs on the end points are equipped with mechanisms to automatically negotiate the duplex mode on a link. However, in particular situations, this auto negotiation procedure might fail [38], resulting in different duplex modes of operation on the involved NICs. This may severely cripple the performance of a network. The work of Shalunov and Carlson [38] describes this problem in detail and provides sufficient information as to construct the belonging FPM that is part of the global FPM for the network. In order to handle Duplex Mismatch problems on links *R9*↔*R10* and *R10*↔*R11*, we instrumented monitoring tools looking for *duplicate acks* and observing the *counters* of the NICs. The resulting incidents were reported to the AFH components and correlated inside the nodes on the basis of the FPM, in order to isolate the right NICs, which are in turn restarted such that duplex auto negotiation is reinitiated.

In addition, the problems of an increased delay on a link and the crash of a DNS server are used in the case study. These problems might occur because of issues in the design and implementation of involved modules, for example 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 AFH 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.

We executed a number of trials with different series of the previously described fault-injections.[f] The ratio of successfully resolved faults is depicted in Figure 4. Thereby, the one axis marks the different trials that were executed in turn, whereas the other shows the absolute number of successful fault-resolutions compared with the number of failed ones. There was one main reason AFH could not remediate all fault-injections, 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 information.

The combined case study presented here shows that indeed, the principles of AFH that are implementable and once provisioned, can enable self-healing features within the network and the devices, thereby improving the quality of an IPv6 network. In addition, the case study also demonstrates how autonomic mechanisms can extend the capabilities of IPv6, for example to overcome ICMPv6 suppression based Back Holes in the core network.

10. SCALABILITY AND OVERHEAD ANALYSIS

The goal of this section is to verify that the 'scalability and overhead' requirement (Req.9 from Section 3) posed on AFH is indeed met by the architecture and the presented prototype. Therefore, in this section, the analysis of the architecture continues by presenting the results from various

[f]In the cases where AFH failed to remove the introduced fault, the automated script for the trial was taking care of this after some time such that the next fault could be introduced.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

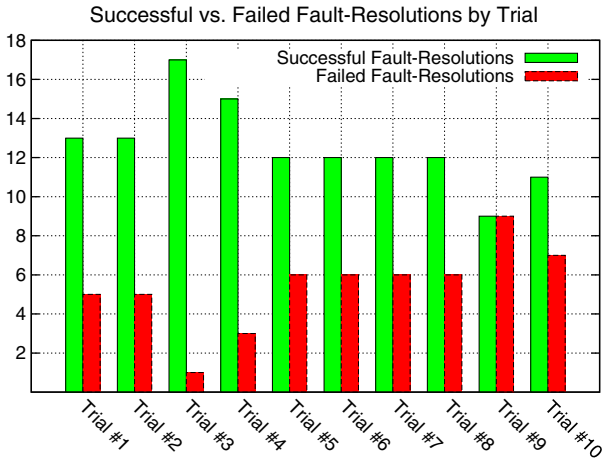

Figure 4. Number of successful Fault-Resolutions during the conducted experiments.

simulations (e.g. OMNET++ simulations) and experiments conducted in the testbed illustrated in Figure 3. By the simulations, single components of the framework are evaluated including the embedded algorithms. Thereby, the inputs to these components are pushed to the extreme in order to gain an impression of performance and overhead.

The distributed control loop 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 that are clearly to be classified as AFH mechanisms are Alarm/Incident-Dissemination, Fault-Isolation, Action Synchronization, Fault-Removal, and other tasks, which depend on our proprietary policy handler. In the following, each of these tasks is analyzed in turn.

10.0.1. Fault-Isolation Functions. The FIF module of AFH is implemented on the basis of the algorithm we developed in our previous work [10]. This algorithm operates on an FPM that, similar with Bayesian Networks, is represented as a directed acyclic graph of events. Thereby, we reformulated the reasoning procedure as to infer the most likely sequence of events on the basis of the detected alarms/incidents, as opposed to Bayesian Networks that infer the probability of each single event to have occurred on the basis of the evidence, that is, the reported alarms/incidents. 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, obtained on an *Intel(R) Core(TM), 2 × '2.80GHz Duo CPUs', 3.9 GB RAM*, are illustrated in Figure 5 and clearly confirm the scalable properties of the FIF module of the AFH prototype. Thereby, Figure 5 shows the 'time required for inference' and the memory consumption of our algorithm as a function of the FPM size on the basis of the presented case study.[g] One can clearly observe that even for quite large models, the reasoning time remains in the magnitude of milliseconds and the consumed memory in the range of several dozens of megabytes.

10.0.2. Incident Dissemination Engine. As previously mentioned, a canonical type of flooding on top of IPv6 was implemented as a first solution for the IDE component of the AFH prototype. Thereby, whenever required, the detecting node sends an alarm/incident message to a list of IPv6 addresses (including multicast addresses as obtained within Multicast-Listener-Discovery). In turn, each of the recipients forwards the message to all IPv6 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 illustrated in Figure 3. In order to evaluate the

[g]The 'number of events' on Figure 5 is based on a small-scale and Fault-Isolation focused simulation of the case study presented in Section 9, and on the growth of the case study network by attaching a new instance of it to one of the edge routers. This network growth results in a growth of the 'number of events' in the Fault-Propagation Model for the simulation.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

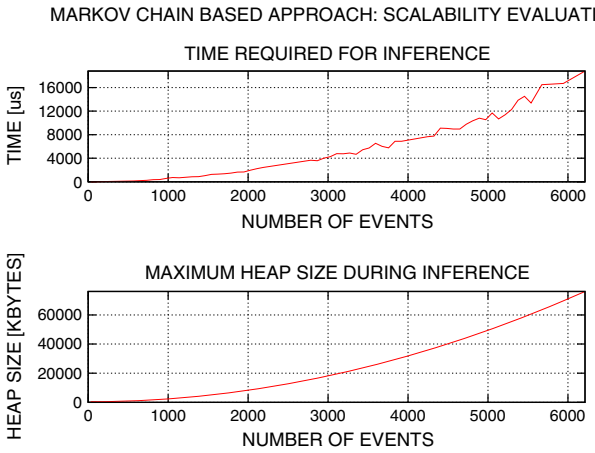

Figure 5. Scalability evaluation of the Fault-Isolation mechanism [10].

Alarm/Incident-Dissemination with respect to its scalability properties, an OMNET++ simulation was developed. Thereby, networks of different sizes (10,100, 200, …, 1000) were simulated on the basis of the Waxman model [39] for Internet topologies. For the generation of these networks, the BRITE tool [40] for random Internet topology generation was used. The experiments compared the implemented flooding procedure with the repetitive retransmission of datagram alarm/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 ms. This resulted in at least a dozen of Giga-Bytes of background traffic for each simulated network. Our findings on the convergence times of the nodes in that setup are visualized in Figure 6. The term ‘convergence time’ denotes the time required for all nodes to receive an alarm/incident description. This ‘convergence time’ is shown on the ordinate, whereas the different network sizes are marked on the abscissa in Figure 6. Figure 6 shows that the flooding procedure achieves pretty low convergence times, whereas 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 implement the IDE and the belonging Alarm/Incident-Dissemination process. However, given the recent advances in the area of information dissemination and message routing [41–43], this topic offers a large potential for applying diverse techniques and investigating their scalability and effectiveness in large networks.

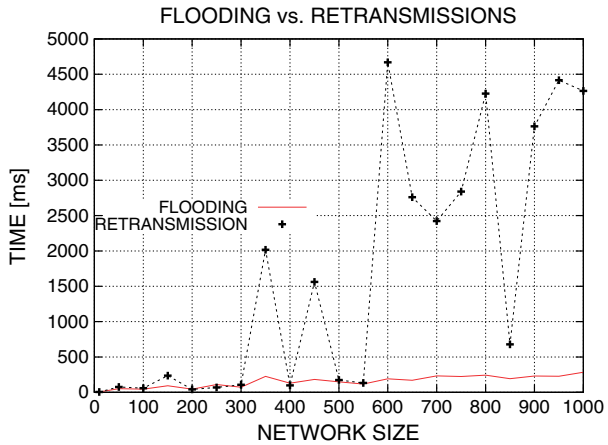

Figure 6. Incident-Dissemination convergence times.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

10.0.3. Policy handler-based components. The AFH framework defines a number of components that were realized on the basis of our proprietary C++ implementation of a policy handler [12] that can operate on XML input. These components are the Fault-Removal Functions, the FRAF, the Asserted Incidents Assessment Functions, and the FIAF. In order to obtain 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 (on an *Intel(R) Core(TM), 2 × '2.80GHz Duo CPUs', 3.9 GB RAM*) is plotted against the policy set size in Figure 7. We see 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.

10.0.4. Action Synchronization Engine. The ASE was implemented on the basis of the techniques presented in our previous work [11]. 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. As previously mentioned, we used an open source solver package [37] for implementing the ASE. The results of our evaluations are presented in Figure 8. In order to evaluate the ASE part of our prototype, we randomly generated 1000 instances for each model size (we took an equal number of potential actions and KPIs that are influenced by these actions). In addition, we used two threads, which simulated the simultaneous issuing of requests (as expected to come from the AFM agent) for all the instances of each model size. The maximum of the response times and the required heap memory (both represented on the ordinate axis), on an *Intel(R) Core(TM), 2 × '2.80GHz Duo CPUs', 3.9 GB RAM*, for each model size (abscissa axis) in the course of these

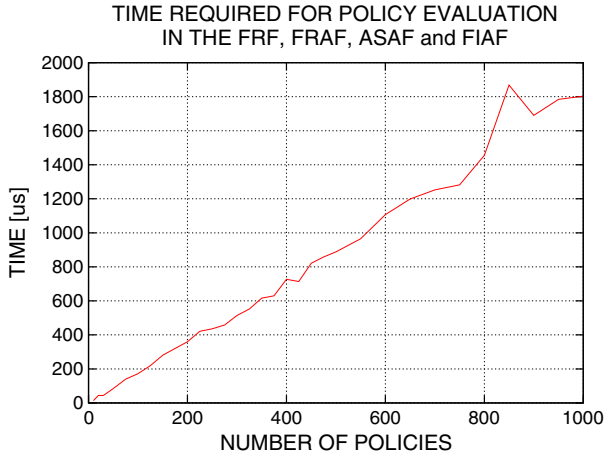

Figure 7. Scalability of the policy handler-based components.

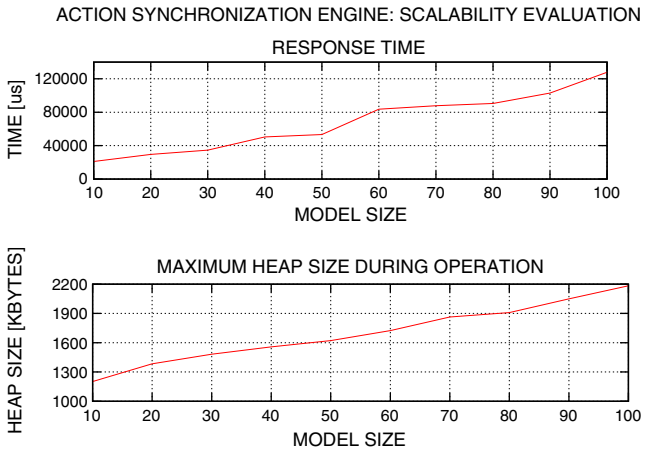

Figure 8. Scalability of the action synchronization engine.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

experiments are presented in Figure 8. These measurements indicate reasonable scalability properties and moderate memory requirements for the ASE part of the AFH prototype.

10.0.5. Scalability and overhead evaluation of the overall distributed control loop. Next, we proceed with an overall evaluation of the distributed control loop. This includes the interplay between the different processes analyzed before, however, without pushing every single process to the extreme as carried out in the previous paragraphs. The experiments were conducted in the testbed described in Figure 3. The scenario for these experiments consisted of the following steps: (i) the process of Fault-Detection was simulated on R4 (a virtual image with *Intel (R) Xeon(R), 'CPU 2.00GHz', 512 MB RAM*) by an entity that periodically pushed incident descriptions to the local set of incident repositories; (ii) the information was pushed to the local AFM agent and forwarded to the IDE on R4; (iii) the IDE on R4 disseminated the incident descriptions across the whole testbed (including routers and end systems); (iv) each router and end system performed Fault-Isolation upon receiving the incident; (v) 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 and executed the *touch* Linux command creating a file locally and emulating the process of Fault-Removal; (vi) the Fault-Removal Assessment was similarly simulated and followed by the dissemination of a Fault-Recovery message across the network; and (vii) 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 previously described procedure was repeated 1000 times in a row in order to gain data regarding the overall execution time of the distributed control loop. These data are plotted in Figure 9 and show that in the current test setup, the execution time for the scenario, simulating the overall distributed control loop, was in the magnitude of several thousands of microseconds. Thereby, the abscissa axis stands for the consecutive trials that were executed, and the ordinate axis shows the time required for the execution of the overall control loop in the scope of the corresponding trial. Finally, additional 50 runs were performed in order to gain experience regarding the AFH memory consumption on the key routers in the scenario, that is, R3 and R4. The results regarding the heap size memory consumption are plotted in Figure 10 against the executed consecutive trials and show again some minimal overhead in the magnitude of dozens of Kbytes.

These measurements related to the overall distributed control loop complement the 'component by component' evaluation of the previous paragraphs, 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.

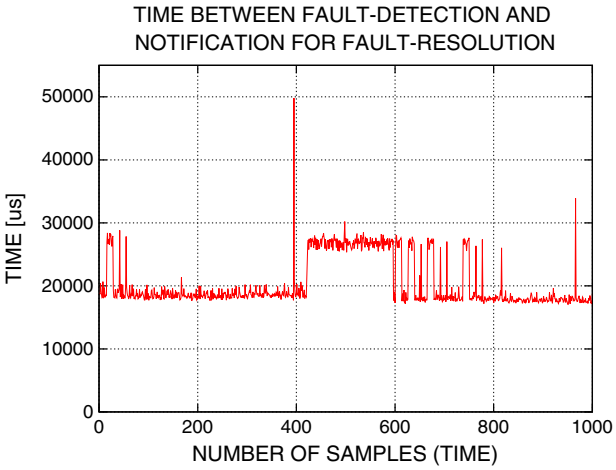

Figure 9. Execution time of the overall AFH control loop.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

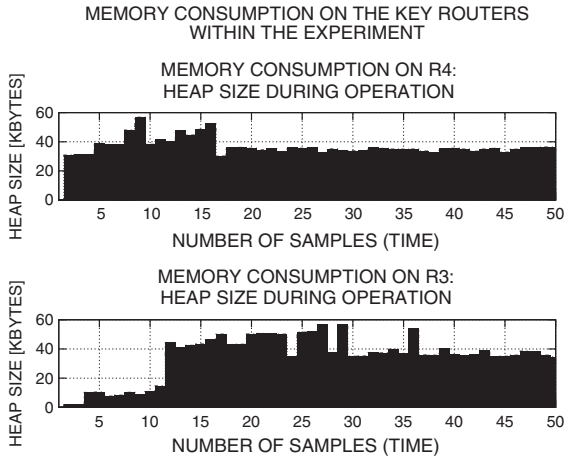

Figure 10. Memory consumption measured during the execution of the overall AFH control loop.

11. CASE STUDY AND RESULTS DISCUSSION

In this section, the key results that were presented in this paper are discussed. The proposed architecture was validated by an implementation that was deployed in an IPv6 environment and used to realize a combined case study involving a number of various networking problems. This case study clearly shows that our proposed architecture is capable of increasing the resilience of an IPv6 network and removing various multiple faults during the operation of the network. The only problem that was encountered was related to the sampling of traffic, that is, to periodical monitoring tasks in general, which could miss to detect corresponding incidents resulting from some networking problems, mainly Black Holes.[h] Therefore, in some cases, the framework failed to detect, isolate, and remove an introduced fault. However, even with this problem, AFH managed to achieve an average fault-resolution rate of 70%, which is quite a substantial benefit for the resilience of an IPv6 network. In addition to the case study, a number of scalability and overhead measurements were provided. These measurements clearly indicate that the framework can operate inside the network devices with minimal overhead when it comes to memory consumption and processing time, and scales good when it comes to its most crucial processes such as Fault-Isolation, Alarm/Incident-Dissemination, Action Synchronization, and Policy Evaluation for the corresponding processes – Fault-Removal, Fault-Removal Assessment, Fault-Isolation Assessment, and so on.

On the basis of the obtained results, a comparison to other self-healing architectures from the domain of autonomic network resilience can be easily conducted. In the past years, a number of excellent efforts in that area were performed. The UniFAFF framework for clean slate networks was implemented in the course of the ANA project [26], and different numerical evaluations of UniFAFF were obtained [25]. However, the scalability of the UniFAFF is inferior as compared with our approach mainly because it builds on an extremely flexible and on-demand configurable ANA [26] kernel, which comes on the price of modest response times and higher memory consumption. Besides, the applicability of the UniFAFF to traditional IP networks has not been evaluated. Sterbenz *et al*. [28] propose the so called $D_2R_2 + DR$ strategy (Defend, Detect, Remediate, Recover, plus Diagnose and Refine), which provides a framework for the implementation of different resilient behaviors [28, 44]. However, these behaviors are implemented on a case by case basis while at the same time complying with the very generic $D_2R_2 + DR$ principles. Hence, in comparison with

[h]Additional experiments with different sampling rates for Black Hole detection on R1 were conducted. These experiments did not show any significant improvement in the Fault-Detection rate after increasing the traffic sampling rate. The authors believe that the limitations are given by the performance of the tools used for traffic monitoring (e.g. tcpdump) and the fact that the traffic monitoring was realized on the forwarding nodes, which resulted in additional overhead in the routers. The Fault-Detection rate has the potential to be improved by using some advanced traffic monitoring technology such as NetFlow, as well as by introducing special link monitoring nodes, which have increased capabilities for observing and analyzing network traffic.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

our approach, there is no single implementation that can be provisioned and configured as to tackle various complex combined network problems. The OMEGA architecture [18, 27] provides a policy-based framework for handling and resolving network faults especially in the context of 3G networks. However, OMEGA does not provide a detailed description of the different phases of a control loop, which we arrived at after the implementation and experimentation with our framework. In addition, OMEGA (as well as the D_2R_2 + DR case study [44]) employs Bayesian Networks to tackle the Fault-Isolation task, which proved to have worse scalability properties than the Markov Chain based algorithm [10] used for AFH. As Fault-Isolation is a key process, this has an enormous impact on the overall scalability of the approach. Furthermore, Vidalenc and Ciavaglia describe in their work [45] a methodology to proactively deal with node/link failures in carrier networks. However, the performance of our framework seems more convincing, which can be explained by the fact that AFH allows to deal with faults more precisely than just on the level of link/node outages. Finally, frameworks of resilience mechanisms based on the migration of virtual machines are in place, for instance [46]. This is of great relevance for the upcoming Internet-based cloud infrastructures. Such mechanisms (e.g. [46]) show some good performance and can be seen as complementary work to AFH, meaning that AFH can embed such mechanisms as part of its fault-resolution strategies. That way, AFH can also play a role in the area of cloud computing.

12. CONCLUSIONS AND FUTURE WORK

This article presented a framework for realizing distributed autonomic self-healing in IPv6 networks. The core of the presented approach is constituted by the idea to introduce components inside the network devices that try to collaboratively resolve faulty conditions in the (long term) operation of the network. Thereby, these components aim at collaboratively realizing the autonomic interplay (without human involvement) of the processes of (i) Fault-Detection – *detect the presence of an erroneous state*, (ii) Fault-Isolation – *identify the root cause(s)*, and (iii) Fault-Removal – *remove the identified root cause(s)*. This interplay is denoted as AFM. The collaboration between the different nodes realizing AFM is enabled by a compartment of alarm/incident dissemination components deployed in each node – IDE. Given that the presence of a faulty condition has been detected by a monitoring sensor in a device, the corresponding alarm/incident description is disseminated across the network. After the network nodes have converged with respect to the alarm/incident information, Fault-Isolation is performed in each device. In turn, some of the nodes react in order to eliminate the root cause(s) on the basis of a pre-configured policy model. The described procedure includes processes that check the quality of the inferred information (Fault-Isolation Assessment) and the executed actions (Fault-Removal Assessment), as well as mechanisms for selecting the optimal set of tentative actions (Action Synchronization). In addition, it is considered that certain challenging situations should be escalated to the network operations personnel in case the problems are not resolvable by means of the distributed control loop structure.

After designing the components at node level, which are meant to realize the aforementioned tasks, the dynamic aspects of the framework were specified. That way, the required interactions among the components within a network node and across the network toward the realization of distributed autonomic self-healing were identified. Moreover, bearing in mind that IPv6 is meant to be the future of networking in the long term, we identify a number of synergies and mutual benefits between the IPv6 protocol suite and our framework. These include (among others) aspects such as security (IPsec), as well as the true end-to-end paradigm and the resulting possibility to involve end systems in the self-healing process. In addition, the presented case study shows how the self-healing processes, realized by the framework, can facilitate IPv6 to overcome Black Hole problems in the core network. These problems arise from the misconfiguration of firewalls suppressing ICMPv6 message types of paramount importance for the correct functioning of the protocol suite across the network.

The proposed framework was validated by various means. An implementation in a testbed was used to apply the principles on a multifaceted case study in an IPv6 environment. The scalability evaluation consisted of the following: (i) an OMNET++ simulation that analyzed a possible realization of the alarm/incident dissemination procedure; (ii) a number of separate simulations (on an

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

Intel(R) Core(TM), 2 × '2.80GHz Duo CPUs', 3.9 GB RAM machine) in which each of the key components was analyzed by performing a number of measurements thereby pushing key evaluation parameters to the extreme; and (iii) a simulation of the whole distributed control loop in the testbed where the prototype was deployed.

The evaluations show that indeed, our framework is implementable and can resolve multiple issues in an IPv6 network thereby improving the resilience of the network by resolving 70% of the injected faults in our case study. This result shows that there is potential for further improving the architecture, processes, and algorithms of AFH especially with respect to Fault-Detection and Fault-Isolation. Furthermore, the presented overhead measurements allow to judge on the scalability of the proposed framework and the overhead it produces inside the network nodes. The indicative measurements show good scalability properties and moderate overhead (e.g. in terms of memory consumption) inside a device (the key measurements within the testbed were centered around a virtual image router with *Intel(R) Xeon(R), 'CPU 2.00GHz', 512 MB RAM*). The overall set of measurements show that AFH has the potential to increase the resilience and robustness of an IPv6 network without imposing the need for major hardware upgrades and thus can bring economic benefits to the network operators.

With respect to future work, there are a number of important challenges that would require further investigation. Firstly, our prototype would need to mature in the course of it being applied to numerous additional case studies. Secondly, user friendly tools are required that would enable the network operations personnel to provide sophisticated models for the functioning of the different involved processes such as Fault-Isolation, Action Synchronization, and Fault-Removal. Finally, the interactions between the network administrator and the self-healing mechanisms can be extended and refined such that diverse *human in the loop* procedures are possible, that is, requesting the human to confirm or reject potential tentative actions of the framework.

ACKNOWLEDGEMENT

This work has been partially supported by EC FP7 EFIPSANS project (INFSO-ICT-215549).

REFERENCES

1. EC funded- FP7-EFIPSANS Project. (Available from: http://efipsans.org/) [as of date 29.12.2012].
2. UNIVERSELF project. (Available from: http://www.univerself-project.eu/) [as of date 29.12.2012].
3. Agoulmine N (ed.). *Autonomic Network Management Principles*. Academic Press, 2010. ISBN: 0123821908
4. Famaey J, Latre S, Strassner J, De Turck F. A hierarchical approach to autonomic network management. Network Operations and Management Symposium Workshops, 2010 IEEE/IFIP, 2010; 225–232.
5. The FCAPS management framework. ITU-T Rec. M. 3400.
6. Bouloutas AT, Calo S, Finkel A. Alarm correlation and fault identification in communication networks. *IEEE Transactions on Communications* 1994; **42**(2–4):523–533.
7. Steinder M, Sethi AS. A survey of fault localization techniques in computer networks. Copyright Elsevier: published in the Journal – *Science of Computer Programming* 2004; **53**:165–194.
8. Raz D, Shavitt Y. Toward efficient distributed network management. *Journal of Network and Systems Management* 2001; **3**:357–361.
9. Tcholtchev N *et al*. Towards a unified architecture for resilience, survivability and autonomic fault-management for self-managing networks. MONA+ workshop, Stockholm, LNCS 6275, September 2010.
10. Tcholtchev N *et al*. Scalable Markov chain based algorithm for fault-isolation in autonomic networks. GLOBECOM 2010; 1–6.
11. Tcholtchev N *et al*. Addressing Stability of Control-Loops in the context of the GANA architecture: "Synchronization of Actions and Policies". Proceedings of IWSOS 2009, LNCS 5918; 262–268.
12. Tcholtchev N, Chaparadza R. Autonomic Fault-Management and resilience from the perspective of the network operation personnel. GLOBECOM Workshops (GC Wkshps), 2010 IEEE; 469–474.
13. Chaparadza R. Requirements for a Generic Autonomic Network Architecture (GANA), suitable for Standardizable Autonomic Behavior Specifications for Diverse Networking Environments. International Engineering Consortium (IEC), Annual Review of Communications, vol. **61**, 2008.
14. Kompella RR *et al*. Detection and localization of network blackholes. Proceedings of IEEE Infocom, Alaska, USA, May 2007.
15. Lakhina A, Crovella M, Diot C. Diagnosing network-wide traffic anomalies. Proceedings of the 2004 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Portland, Oregon, USA, August 30-September 03, 2004.

Copyright © 2013 John Wiley & Sons, Ltd.

DOI: 10.1002/dac

16. Tang Y, Al-Shaer E, Boutaba R. Efficient fault diagnosis using incremental alarm correlation and active investigation for internet and overlay networks. *IEEE Transactions on Network and Service Management* 2008; **5**(1):36–49.
17. Hasan M, Sugla B, Viswanathan R. A conceptual framework for network management event correlation and filtering systems. In *Integrated Network Management VI*, Sloman M, Mazumdar S, Lupu E (eds.). IEEE, Boston, MA, 1999; 233–246.
18. Baliosian J *et al*. The Omega Architecture: towards adaptable, self-managed networks. Annual Workshop on Distributed Autonomous Network Management Systems, Dublin, Ireland, June 2006.
19. Varga P, Moldovan L. Integration of service-level monitoring with fault management for end-to-end multi-provider ethernet services. *IEEE Transactions on Network and Service Management* 2007; **4**(1):28–38.
20. Sterritt R *et al*. Exploring autonomic options in an unified fault management architecture through reflex reactions via pulse monitoring. 11th IEEE International Conference and Workshop of Engineering of Computer-Based Systems, 24-27 May 2004; 449–455.
21. Ekaette EU, Far BH. A framework for network fault management using software agents. *IEICE Transactions on Information and Systems* 2004; **E87-D**(4):947–958.
22. Ekaette EU, Far BH. A framework for distributed fault management using intelligent software agents. Electrical and Computer Engineering, 2003, IEEE CCECE 2003, Canadian Conference on, vol. 2, 4–7 May 2003; 797–800.
23. Li N, Chen G, Zhao M. Autonomic Fault Management for wireless mesh networks. *Electronic Journal for E-Commerce Tools and Applications (eJETA)* 2009; **2**(4):1–8.
24. Chaparadza R. UniFAFF: a unified framework for implementing autonomic fault-management and failure-detection for self-managing networks. *International Journal of Network Management* 2008; **19**(4):271–290, July/August 2009.
25. Chaparadza R, Tcholtchev N. Implementation of the UniFAFF framework for Autonomic Fault-Management in ANA Networks. ICLAN'2008, Toulouse France, 10-12/12/2008.
26. ANA project. (Available from: http://www.ana-project.org/) [as of date 29.12.2012].
27. Baliosian J *et al*. Policy-based self-healing for radio access networks. In Network Operations and Management Symposium, 2008, NOMS 2008, IEEE, April 2008; 1007–1010.
28. Sterbenz JPG *et al*. Resilience and survivability in communication networks: strategies, principles, and survey of disciplines. *Computer Networks* 2010; **54**(8):1245–1265.
29. Zhou X, Jacobsson M, Uijterwaal H, Van Mieghem P. IPv6 delay and loss performance evolution. *International Journal of Communication Systems* 2008; **21**(6):643–663.
30. Chen W-E, Lin P-J. A performance study for IPv4–IPv6 translation in IP multimedia core network subsystem. *International Journal of Communication Systems* 2010; **23**(8):929–944.
31. Grajzer M, Żernicki T, Głąbowski M. ND++ – an extended IPv6 Neighbor Discovery protocol for enhanced stateless address autoconfiguration in MANETs. *International Journal of Communication Systems* 2012. DOI:10.1002/dac.2472.
32. Oliveira LML, de Sousa AF, Rodrigues JJPC. Routing and mobility approaches in IPv6 over LoWPAN mesh networks. *International Journal of Communication Systems* 2011; **24**(11):1445–1466.
33. Wang X, Qian H. Hierarchical and low-power IPv6 address configuration for wireless sensor networks. *International Journal of Communication Systems* 2012; **25**(12):1513–1529.
34. Le A, Loo J, Lasebae A, Aiash M, Luo Y. 6LoWPAN: a study on QoS security threats and countermeasures using intrusion detection system approach. *International Journal of Communication Systems* 2012; **25**(9):1189–1212.
35. Shi L, Davy A. Security considerations for intrinsic monitoring within IPv6 networks. Published in the Proceedings of the 9th IEEE IPOM Workshop, Venice, Italy, October 2009.
36. OMNET++ network simulation framework. (Available from: http://www.omnetpp.org/) [as of date 29.12.2012].
37. Coin-OR. (Available from: http://www.coin-or.org/) [as of date 29.12.2012].
38. Shalunov S, Carlson R. Detecting duplex mismatch on ethernet. Passive and Active Network Measurement, 6th International Workshop, PAM 2005, Boston, MA, USA, March 31–April 1, 2005.
39. Zegura EW. How to model an internetwork. *Proceedings IEEE INFOCOM '96* 1996; **2**:594–602.
40. BRITE. (Available from: http://www.cs.bu.edu/brite) [as of date 29.12.2012].
41. Tang J, Dai S, Li J, Li S. Gossip-based scalable directed diffusion for wireless sensor networks. *International Journal of Communication Systems* 2011; **24**(11):1418–1430.
42. Busson A. Analysis and simulation of a message dissemination algorithm for VANET. *International Journal of Communication Systems* 2011; **24**(9):1212–1229.
43. Wang X. Analysis and design of a k-Anycast communication model in IPv6. *Elsevier Computer Communications* 2008; **31**(10):2071–2077.
44. Marnerides AK *et al*. Remediating anomalous traffic behaviour in future networked environments. Access Networks 2010, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Volume **63**, Part 7, 2011; 187–197.
45. Vidalenc B, Ciavaglia L. Proactive fault management based on risk-augmented routing. GLOBECOM Workshops (GC Wkshps), 2010 IEEE, 6–10 Dec. 2010; 481–485.
46. Fischer A, Fessi A, Carle G, de Meer H. Wide-area virtual machine migration as resilience mechanism. Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on, Oct. 2011; 72–77.

Copyright © 2013 John Wiley & Sons, Ltd.