Technische Universität München

Fakultät für Informatik

Lehrstuhl für Netzarchitekturen und Netzdienste

Adaptive and Safe Network Configurations in Safety-Critical Systems

Cora Lisa Perner

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades einer

Doktorin der Ingenieurwissenschaften

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Darius Burschka

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Georg Carle
2. Prof. Dr.-Ing. Manfred Hajek

Die Dissertation wurde am 08.04.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 06.10.2021 angenommen.

Abstract

Systems are considered to be safety-critical if their failure could cause serious damage to the environment or injuries. When networks are in use in such systems such as the electricity grid or on-board vehicular networks, the relevant policies and routing strategies are hard-coded on network switches. Beyond that, they rely on additional hard-wired redundancy for safety (i.e. ensuring that no serious damage occurs) and restricted physical access for security (i.e. preventing unauthorised access). However, this approach is inflexible, complex to maintain and the implementation may vary between vendors. In addition, an increasing interconnection of safety-critical systems with the internet for convenience and added functionalities has increased the exposure to cyber risks that cannot be addressed to date.

Emerging technology such as software-defined networking (SDN) would allow to address these shortcomings by providing means to react dynamically to detected deviations from normal (i.e. safe and secure) behaviour, or anomalies. Yet as this technology has not been designed for safety-critical systems, provisions need to be made to ensure that safety-critical functionalities can be guaranteed. However, dynamic and best-effort strategies are not acceptable for critical system as they would result in unpredictable behaviour especially with respect to availability and delay bounds.

Consequently, this thesis advocates the use of pre-calculated network configurations that can be applied to react to a detected anomaly. Such *configuration templates* can guarantee that all requirements of safety-critical traffic are satisfied.

Several different algorithms – both heuristic (Dijkstra) and optimal (minimum cost, minimum link utilisation, minimum queueing delay, minimum number of forwarding table entries, and maximum resilience) – were applied on several different network topologies to obtain such templates, and their effect on network performance investigated. As calculations are performed prior to deployment, it can be verified whether all requirements of safety-critical traffic are ensured. However, it was found that Dijkstra-based heuristics – even if they account for the delay bound – result in violations of requirements. In addition, it was found that optimisation for minimum cost provides good results for the performance characteristics investigated. In contrast, the benefits for minimising queueing delay, link capacity or forwarding tables were minimal, and do not justify the significant increase in calculation time.

In addition, transmission of safety-critical traffic may not be interrupted. Consequently, the processes necessary to transfer a configuration template into a running configuration have been studied. Using two both node- and link-disjoint paths concurrently, the path affected by a failure can safely be reconfigured without affecting safety-critical traffic. In case of a component failure, one of those paths can be reconfigured, thus allowing the safety-critical transmission to continue without interruption. Moreover, as up to nine independent paths can be obtained using the maximum resilience, further failures can be tolerated without the need to enter a degraded mode. This would allow e.g. aircraft to continue the journey to their destination rather than needing an emergency landing.

Beyond that, the effect of using templates in order to react to network anomalies has been investigated. To this end, up to ten failures of network components were randomly triggered in the network and the effect on the success of the reconfiguration was studied. It was found that the number of necessary reconfigurations decreases with increasing failures, as isolating traffic sources and sinks reduces the traffic to be rerouted. For the same reason, failures of switches have a more significant impact on network traffic than link failures.

In addition to network reconfiguration (which may not always be a suitable reaction), this thesis also presents a general control-system-based approach to react to anomalies, such as buffer overflows. Here, the most important feature is that it is ensured that the impact of the reaction is proportional and appropriate to the detected anomaly. This ensures that minor anomalies (such as e.g. a slight increase in the average queueing delay) do not cause a reduction of safety margins. Moreover, it may be the case that some anomalies or intrusions cannot be mitigated as such an effort would likewise result in a reduction of safety margins. In safety-critical systems, security is a means to achieve safety, but the latter always needs to come first.

Zusammenfassung

Systeme werden als sicherheitskritisch eingestuft, wenn ihr Ausfall zu schweren Umweltschäden oder Verletzungen führen könnte. Beim Einsatz von Netzwerktechnologien im Stromnetz oder in der Fahrzeugvernetzung werden Richtlinien und Routing-Strategien verwendet, die auf Netzwerkswitches fest codiert sind. Darüber hinaus sind sie für die Betriebssicherheit (*safety*) auf zusätzliche, fest verdrahtete Redundanz und zum Schutz vor unberechtigtem Zugriff (*security*) auf eingeschränkten physischen Zugang angewiesen. Dieser Ansatz bringt Nachteile wie geringe Flexibilität, hohe Komplexität bei der Wartung und fehlende Harmonisierung bei der Implementierung mit sich. Des Weiteren hat die zunehmende Vernetzung von sicherheitskritischen Systemen mit dem Internet zur Erweiterung von Funktionalitäten und User Experience dazu geführt, dass diese höheren Cyber-Risiken ausgesetzt sind.

Aufkommende Technologien wie software-defined networking (SDN) würden es ermöglichen, diese Nachteile zu beheben, indem sie Methoden zur Verfügung stellen, dynamisch auf erkannte Anomalien, also Abweichungen von sicherem Verhalten (sowohl in Hinblick auf Betriebssicherheit als auch zum Schutz vor unberechtigtem Zugriff) zu reagieren. Da diese Technologie jedoch nicht für sicherheitskritische Systeme konzipiert wurde, müssen Maßnahmen getroffen werden, um sicherheitskritische Funktionalitäten gewährleisten zu können. Dynamische bzw. Best-Effort Strategien sind jedoch für kritische Systeme nicht akzeptabel, da sie zu unvorhersehbarem Verhalten führen würden – insbesondere im Hinblick auf Verfügbarkeit und Latenz-Obergrenzen.

Diese Arbeit beschreibt eine Vorgehensweise, die auf erkannte Anomalien mit im Vornherein ermittelte Netzwerkkonfigurationen reagiert. Solche *Konfigurationstemplates* ermöglichen es, dass alle Anforderungen von sicherheitskritischem Netzwerkverkehr erfüllt werden.

Hierzu untersucht diese Arbeit die Anwendung verschiedene Algorithmen – sowohl heuristische als auch optimale (minimale Kosten, minimale Auslastung von Links, minimaler Delay, minimale Anzahl von Einträgen in Routingtabellen und maximale Resilienz) – auf verschiedene Netzwerktopologien zur Ermittlung der Templates. Anschließend kann die resultierende Netzwerkperformance analysiert werden. Eine Berechnung und Bewertung der Templates findet vor Inbetriebnahme statt, sodass sichergestellt werden kann, dass alle Anforderungen an einen sicherheitskritischen Nachrichtenaustausch erfüllt werden. Bei den auf Dijkstra basierenden Heuristiken lässt sich feststellen, dass selbst unter Berücksichtigung von Latenzbeschränkungen die Anforderungen von sicherheitskritischen Systemen verletzt werden. Die Optimierung auf minimalen Kosten stellte sich unter Betrachtung der untersuchten Parameter als ein geeignetes Verfahren dar. Im Gegensatz dazu sind die Vorteile einer Minimierung des Delays, Einträgen in Routing-Tabellen oder Link-Kapazität minimal, und rechtfertigen nicht den erheblich größeren Aufwand an Rechenzeit.

Darüber hinaus darf die Übertragung von sicherheitskritischem Verkehr nicht unterbrochen werden. Folglich wurden die Prozesse, um ein Konfigurationstemplate in ein lauffähiges Template überzuführen, untersucht. Durch die gleichzeitige Verwendung von zwei knoten- und linkdisjunkten Pfaden kann der von einem Ausfall betroffene Pfad sicher rekonfiguriert werden, ohne den sicherheitskritischen Netzwerkverkehr zu beeinträchtigen. Im Falle eines Komponentenaus-

falls kann einer dieser Pfade rekonfiguriert werden, sodass die sicherheitskritische Übertragung ohne Unterbrechung fortgesetzt werden kann. Zudem erlaubt dieses Vorgehen folgende Ausfälle zu tolerieren, weil bis zu neun unabhängige Pfade unter Ausnutzung der maximalen Kapazität verwendet werden können, ohne dass das gesamte Netzwerk in einen degradierten Modus geschaltet werden müsste. Dies würde es beispielsweise Flugzeugen ermöglichen, die Reise zu ihrem Zielort fortzusetzen, anstatt eine Notlandung durchführen zu müssen.

Außerdem ergibt sich die Möglichkeit der Verwendung von Templates zur Reaktion auf Netzwerkanomalien. Zu diesem Zweck sind bis zu vierzehn Ausfälle von Netzwerkkomponenten zufällig ins Netzwerk eingebracht und die Auswirkung auf eine erfolgreiche Rekonfiguration untersucht worden. Es lässt sich feststellen, dass die Anzahl der notwendigen Rekonfigurationen mit zunehmenden Ausfällen abnimmt, weil die Isolierung von Nachrichtenquellen und -senken den umzuleitenden Verkehr verringert. Aus dem gleichen Grund haben Ausfälle von Switches einen größeren Einfluss auf den Nachrichtenaustausch als Verbindungsausfälle.

Neben der Rekonfiguration des Netzwerks (die nicht immer eine geeignete Reaktion sein muss) wird in dieser Arbeit auch ein allgemeiner regelkreisbasierter Ansatz vorgestellt, um auf Anomalien, wie etwa Buffer Overflow, zu reagieren. Dabei ist das wichtigste Merkmal die Sicherstellung, dass die Auswirkung der Reaktion angemessen zu der festgestellten Anomalie ist. Dadurch wird sichergestellt, dass kleinere Anomalien (wie etwa eine leichte Erhöhung des durchschnittlichen Queueing-Delays) nicht zu einer Verringerung der Sicherheitsmargen führen. Darüber hinaus gibt es Fälle, in denen einige Anomalien oder Eingriffe nicht abgemildert werden können, weil solche Maßnahmen ebenfalls zu einer Verringerung der Sicherheitsreserven führen würden. In sicherheitskritischen Systemen ist Security ein Mittel, um Betriebssicherheit zu erreichen, aber letztere muss immer an erster Stelle stehen.

Acknowledgements

First and foremost, I would like to extend my deepest gratitude to all members (past and present) of the Chair of Network Architecture and Services for sharing their extensive knowledge. I must specifically thank Holger Kinkelin and Marcel von Maltitz for their insightful theoretical and practical suggestions with conducting a research project.

Equally, I am extremely grateful to my colleagues at Airbus for their valuable advice and regular discussions in the coffee corner helping me grow as a scientist and a person. I also wish to thank my employer and the German Ministry of Science and Research for the beneficence that enabled the research that led to this thesis.

Beyond this, I cannot begin to express my thanks to Sascha Uhrig, who took on the role of my mentor, for constructive and extremely helpful feedback (and the occasional nudge to start publishing). Moreover, I would like to extend my gratitude to Markus Klügel for his constructive and extremely helpful feedback. I must also thank Clemens Moeser and Dominic Schupke for their practical suggestions and support. I'd like to acknowledge the help of Fabien Geyer in the early phases of my PhD research project.

Last but by no means least, I must thank my parents for encouraging and nurturing my love for science, technology, and engineering. I would also like to extend my sincere thanks to my family and partner for their love, support, patience, and (most importantly) cooking. This thesis would not have been possible without you!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Networks currently used to support the functionalities of safety-critical systems such as airplanes or the power grid employ decentralised network management. Here, each network switch implements policies and traffic routing. Furthermore, it needs to have knowledge of the entire network. Consequently, any changes to the network e.g. due to new components or traffic demands, require adaptations to every single switch. While protocols exist to partially automate such tasks, those are vendor-dependent [22], thus combining components of different vendors or even different hardware standards is challenging and often impossible.

Hence, this approach is complex and expensive to maintain, extend and adapt to new technologies. Consequently, such networks can only support limited complexity and network sizes. This is especially relevant given that safety-critical systems may remain in operation for several decades and need to be able to adapt to significant technological changes.

In addition to those practical considerations, information systems commonly demand *availability, confidentiality, and integrity* [101]. Beyond that, safety critical systems require additional guarantees. These requirements derive from the underlying certification demand that no single fault may result in a failure with catastrophic results [48, S.2-F-47]. Additionally, some traffic (e.g. a braking signal) needs to arrive at its destination within a given time.

Hence, for each safety-critical demand, the following conditions need to be satisfied:

1. Availability
2. Resilience against faults and failures
3. Timing guarantees/predictability for real-time systems

While availability is required for both safety-critical and information systems, confidentiality has not traditionally been a key feature. On the contrary, for many systems broadcasting data is an important safety feature, e.g. broadcasting clearances to all aircraft so that contradictions can be spotted more easily. However, with an increasing interconnection e.g. in the scope of the smart grid, preserving privacy becomes increasingly important. Finally, data integrity is relevant for most critical systems, yet previously physical access to the system in question was required. Consequently, physical security measures sufficed to ensure data integrity. This is no longer the case in interconnected systems.

Hitherto, some safety-critical systems e.g. airplanes rely on hard-wired physical redundancy to ensure resilience against faults and failures. These additional paths are separated from the main network and bypass it completely. While this approach provides resilience, it also increases weight and complexity. An increase in weight is problematic for vehicular networks, as an increase in mass also increases fuel consumption. Furthermore, given the same failure probability, more components make it more likely that any one failure occurs. In addition, a more complex system is harder to analyse, thus making it more likely for a critical defect to go unnoticed.

Yet resilience is not only the capacity to withstand unintentional faults (e.g. due to ageing components), but also intentional ones. Previously, security for critical systems has been mainly considered in the context of restricting physical access to critical components as they were not connected to the internet. Yet this approach has been called into question when Stuxnet has been uncovered in 2010 [92]. This malicious worm infected Siemens SP7 programmable logic controllers (PLCs) used in a plethora of supervisory control and data acquisition (SCADA) systems and caused significant physical damage to the Iran's nuclear enrichment centrifuges, despite being not directly connected to the Internet. While this infection of PLCs was widely assumed to be a targeted attack by nation state actors, it raised the awareness of operators that even air-bridged systems may be susceptible to cyber-infections and attacks.

Yet only a small percentage of network traffic is actually safety-critical, with a larger quantity related to diagnostics, convenience and management. Notwithstanding, traffic flows with lower requirements obviously influence those with higher demands (e.g. by potentially congestion of the network, starving critical traffic of resources etc.).

Consequently, safety-critical systems are increasingly (and intentionally) connected to the Internet for added functionality or convenience. For example, under the blanket term 'smart grid', digital meters are introduced to facilitate billing and management of power stations. Furthermore, an increasing number of airlines and car manufacturers provide on-board Internet access to their passengers, albeit for a premium. Together with the increased use of commercial off-the shelf (COTS) components that are not specifically hardened for the use in critical systems, considering network security becomes increasingly important. Yet due to the safety-critical application, such networks need to be able to continue to operate even when faced with threats. Relying on human administrators is unpractical, as they might take too long to react to an event, if they are present at all. Consequently, continued safe operation needs to be ensured by automatically reacting to failures.

Yet currently used technology does not support such dynamic network adaptation in response to the detection of a failure or attack. Instead, an emphasis is placed on hard-wired physical redundancy and methods such as network calculus [106] to ensure that critical traffic can achieve its safety goals. These methods are used to achieve performance goals and to ensure that the critical functionality (e.g. providing control commands to actuators for braking) can be performed safely. Additionally, critical information flows are replicated k times and transported via different routes across the network.

This leads to an increasing interest in mechanisms such as software-defined networking (SDN) (cf. section 2.3). SDN allows to separate the handling of traffic from network management. Thus, network management can be more flexible and effective [103]. In addition to the capabilities of today's networks, SDN permits to change the network configuration during runtime by redirecting traffic flows. This additional flexibility could be used to adaptively react to changes in the network that are caused by faulty components as well as network security incidents. Consequently, interesting prospects arise with respect to flexibility and better overall performance.

For example, if a SDN network switch is targeted by an attack, the respective node can be isolated by rerouting traffic to avoid compromising further components. Likewise, traffic could be temporarily redirected while an overheated component cools down. This adaptability could allow to tolerate more failures safely without the need to enter a degraded mode or postpone the need for a maintenance action. Hence, an operation could be continued safely until a fault or vulnerability can be repaired, e.g. allowing a data centre to continue autonomous operation until a technician can reach the affected component or a bug fix is issued by the manufacturer.

Due to these benefits, SDN is receiving some attention in areas for which it was not originally designed: for safety-critical applications. Yet if a network for safety-critical applications can be made sufficiently reliable through such means, additional hard-wired links may safely be removed. This would allow to reduce weight – highly relevant for networks on mobile platforms such as aircraft – and complexity. As SDN it is also one of the key features introduced with upcoming 5G standard for telecommunication networks, it can also be used to facilitate vehicle to infrastructure (V2X) communication for localisation, trajectory planning, or traffic flow control.

Network optimisation provides a means to incorporate these demands of critical systems while taking the limitations of SDN into account. Current network optimisation efforts e.g. [119, 139, 161, 147] focus mainly on balancing network loads. However, for safety-critical systems a different approach is needed. Here, the demands for network management are based on the functionalities mentioned above rather than performance issues. Notwithstanding, the limitations of network capacity still apply.

Satisfying the relevant constraints can be ensured by using network optimisation or network calculus. Yet without a suitable objective function the resulting network might still not yield the best possible performance for various metrics. While different optimisation goals result in varying network parameters, the effect of different objective functions for critical systems on network performance has to date not been studied and compared in detail.

1.1 Research Questions

Currently used networks are static and unable to respond to emerging challenges. While SDN would enable dynamic network changes, it cannot currently be used for safety-critical systems, as the requirements of safety-critical traffic are not taken into account. In addition, there has been no comprehensive comparison of the impact of using different objective functions and their impact on network performance and safety-critical traffic. Consequently, the main research question addressed in this thesis is

How can safety-critical networks react dynamically and safely to failures?

This research question can be subdivided into two parts: how to obtain candidate responses (configuration templates) and how to apply these responses safely (network configurations) to address detected anomalies. This division is reflected in the structure of this chapter and henceforth this thesis: section 1.2.1 further subdivides the research question w.r.t. template generation, while section 1.2.2 details further issues of how reactions to failures can be applied. The relationship between the work covered in this thesis and the Open Systems Interconnection (OSI) model is depicted in figure 1.1. Here, the highlighted layers indicate the focus of this thesis.

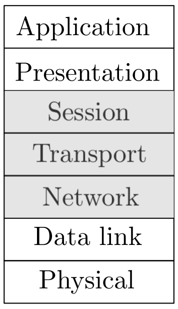


Figure 1.1: Relationship between this thesis and the OSI model

1.2 Structure of This Thesis

To begin with, an overall system model is presented in chapter 3. This chapter also introduces details when a reaction can be considered to be *appropriate* to the detected anomaly. As the research question itself, the remainder of this thesis is divided into two main parts that will be detailed in the following paragraphs.

1.2.1 Generating Configuration Templates

In order to generate network configuration templates, several steps are needed. To begin with, it is necessary to translate safety requirements into network-level requirements, as described in section 3.3. Consequently, desired network behaviour can be determined and thus be the foundation for future evaluation. Subsequently, in order to find the most suitable algorithm(s) and objective function for safety-critical systems, the effect thereof on the network and the traffic will be studied in chapter 4. To this end, several different network topologies are investigated and the results with respect to network performance characteristics as well as calculation performance will be detailed.

In this context, it is also important to consider the resources required to obtain network configurations to establish whether the algorithm/method can be deployed online or needs to be calculated beforehand, which will be discussed in detail in section 4.6.

1.2.2 Applying Configurations

Once those basic parameters and configuration templates have been established for the data plane, the behaviour of network during the reconfiguration will be studied in chapter 5. This is one of the key questions for deployment in safety-critical systems, as safety and related constraints need to be ensured for the entire operation.

Consequently, it is necessary to study the impact of the reconfiguration on the management plane by studying the required time and the amount of control traffic exchanged. This allows to determine whether a separate control network is required or whether control can be in-band. Furthermore, in order to assess the performance of the reconfiguration, it is necessary to time the reconfiguration. In addition, to investigate the effectiveness of the studied means to increase resilience, the impact of successive failures need to be studied.

Beyond those performance indicators, the two major considerations with respect to network reconfigurations are safety and security. With respect to safety, several characteristics are of importance. Firstly, the reaction process needs to be accountable in order to ensure that any (potentially) unsafe conditions can be rectified in a timely manner. Secondly, the reaction needs to be appropriate to the detected anomaly to ensure that minor network issues do not have a significant impact on safety-critical traffic. Consequently, it is also necessary to analyse the impact of the reaction on network performance. Finally, it needs to be established whether the reaction has mitigated the effect of the anomaly.

In this thesis, security is considered as a means to achieve safety. Notwithstanding, some security properties need to be considered in the context of network configuration, as described in section 5.1. To begin with, malicious network configurations need to be prevented, as they could result in the loss of safety-critical traffic. As SDN centralises control, it needs to be ensured that using this process does not introduce an additional attack path. Finally, the effect on security on the data, control and management plane will be investigated in section 5.2. While the main focus of those considerations is the management plane, diverging considerations for data and control plane are made where applicable.

1.2.3 Conclusion

This thesis concludes with remarks as well as an outlook to potential future research directions. Here, chapter 6 provides a discussion of the results and methods used in this thesis. Subsequently, a summary of the main findings of this thesis is given in chapter 7, which includes details on possible ways as to how to build upon this thesis in future research in section 7.4.

1.3 Publications in the Context of This Thesis

Some of the findings of this thesis have already been published as listed below. Specific references will be made at the beginning of the respective chapter/section.

C. Perner. Network Optimization for Safety-Critical Systems Using Software- Defined Networks. In: *Architecture Comput. Sys. (ARCS) 2018*. Ed. by Mladen Berekovic et al. Cham: Springer International Publishing, 2018, pp. 127–138. isbn: 978-3-319-77610-1.

C. Perner and G. Carle. Comparison of Optimization Goals for Resilient Routing. In: *2019 IEEE International Conference on Communications Workshops (ICC Workshops): The 2nd International Workshop on 5G and Cooperative Autonomous Driving (5G Auto) (ICC 2019 Workshop - 5G Auto)*. Shanghai, P.R. China, May 2019.

C. Perner, H. Kinkelin, and G. Carle. Adaptive Network Management for Safety-Critical Systems. In: *IM 2019 - IEEE/IFIP Workshop Dissect 2019*. Washington D.C., USA, Apr. 2019.

C. Perner, C. Schmitt, and G. Carle. Dynamic Network Reconfiguration in Safety-Critical Aeronautical Systems. In *39th AIAA/IEEE Digital Avionics Systems Conference (DASC)*. San Antonio, USA, Oct. 2020.

Chapter 2

Background

This chapter describes background information relevant for the aspects covered in this thesis. Following the definition of the key terms used in this thesis in section 2.1, the requirements imposed on safety-critical systems by regulators as well as other relevant parties will be presented in section 2.2, which will also include a discussion on accountability. Subsequently, the concept of SDN will be presented in section 2.3 including a brief overview of previous efforts to use SDN in critical systems. Furthermore, section 2.4 will describe the key components to ensure cyber security, both for general cyber-physical systems and specifically for those using SDN. This chapter concludes with a brief summary in section 2.5.

2.1 Terminology

To begin with, the key terms in this thesis will be explained: faults, failures, resilience, cyber-physical systems and safety-critical systems.

2.1.1 Faults & Failures

In this thesis, the following definitions are used. In [83], a *fault* is defined as

> (...) an unpermitted deviation of at least one characteristic property (feature) of a system from the acceptable, usual, standard condition,

while a *failure* is specified as

> (...) permanent interruption of a system's ability to perform a required function (...)

Consequently, *fault tolerance* is the ability of a system to tolerate faults so that they do not result in a system's failure, while *resilience* is a system's ability to continue operation despite the presence of failures. Here, *redundancy* i.e. implementing the same function several times can be used to prevent the loss of a function. Further details on this aspect, together with the closely related concept of resilience are given below.

Finally, *survivability* is the ability of a system to timely fulfil its mission in the presence of attacks or large-scale natural disasters [167].

2.1.2 Cyber-Physical Systems

In this thesis, a cyber-physical system is a system, in which a cyber system has an impact on a real-world, physical system. An example would be a computer-controlled braking system.

2.1.3 Safety-Critical System

Beyond that, a safety-critical system is a system whose failure can have a catastrophic impact on humans, animals or the environment and results in serious or lethal injuries.

2.2 Requirements

For the purpose of this thesis, the requirements of three types of safety-critical systems shall be analysed: aircraft, automotive, and the smart grid. As the aircraft sector is highly regulated [48], the most detailed list of applicable regulations can be obtained from the relevant regulatory authorities. Yet considering the requirements for automotive is of equal importance. Firstly, because there are far more automobiles than aircraft. Secondly, the automotive case has privacy implications that do not occur in airplanes, as individual movement patterns are far harder to determine. Finally, due to the extremely large networks, considering the requirements of the smart grid are equally relevant. Moreover, similar requirements with respect to privacy are relevant in this scenario. In addition, much more people will be affected by a malfunction in the electricity grid compared to the other two sectors. However, shared requirements will not be repeated.

2.2.1 Aeronautics

Aviation is (amongst others) highly vulnerable to malicious attacks as it is not separable from global communication through satellite links [43], even more so through the adoption of open/common standards (e.g. Ethernet) for data networks [172]. In principle, security threats can be divided into two subtypes: external (mainly via satellite or wireless) and internal (i.e. from passengers, crew and maintenance personnel that have access from within the aircraft).

An emerging challenge for aircraft security is the interconnectedness of various systems, which in turn offers new attack paths. For example, an attacker may infiltrate air traffic control (ATC) and then continue on to aircraft [99].

Another challenge for aircraft security is the long life-cycle of products. Aircraft are expected to stay in service for well over 20 years, so a static security will result in a deterioration of effective security levels over time. Equally, it is not possible from an operational point of view to ground an aircraft to wait for a patch to a known vulnerability, while on the other hand, airworthiness authorities will not allow an aircraft to take off with known vulnerability [99]. Further, software needs to be designed so that upgrades do not require re-certification of the entire software [179]. In particular, non-safety critical systems must impact safety-critical functions.

In addition to ensuring the safety of the aircraft and its occupants, a data network also has three basic security requirements: confidentiality (ensures privacy of end users), authentication (controls access to network resources) and integrity. However, for aircraft, network security is more important than application security [6].

In [99], the authors advocate that a designer for an aeronautical security application should always consider the (potential) impact on all stakeholders. Equally, security should not be based exclusively on protection from harm but rather use the concept of integrated security: identifying all operational requirements and interdependencies.

With respect to internal threats, an important consideration is that an attacker does not need direct access to flight controls to create a safety risk [99]. It would suffice to create passenger panic e.g. by displaying threatening messages on the in-flight entertainment (IFE) displays.

According to [99], classical intrusion detection systems (IDSs) are not useful to implement on aircraft, as there is no-one on board to alert and select a suitable reaction. Pilots are not suitable for three reasons:

1. They are busy flying the aircraft
2. They have no detailed knowledge in computer security
3. They do not know how to respond

On the other hand, broadcasting every alert to the ground is also unsuitable, as a knowledgeable attacker could easily jam the ground link. However, using autonomous decision making methods on board would be able to mitigate these issues.

In a position paper [81] , the International Federation of Air Line Pilots' Associations (IFALPA) recommends that updates to software systems are protected against both internal and external threats, and that security issues are resolved by updates shortly after they become known. However, as aircraft are highly complex, safety-critical systems, it is necessary to prove (not least to airworthiness authorities) that the update does not interfere with the correct operation of the system. Furthermore, the trade body representing airlines [80] states that the likelihood of cyber-threats to the global aviation systems is increasing, partly due to the integration of systems and processes.

Equally, the Advisory Council for Aeronautics Research in Europe (ACARE) [3] demands that aircraft are able to neutralise security threats to the aircraft's systems and that they are - by design - resilient to both current and predicted threat evolution. What is more, [3] also requires the entire air traffic management (ATM) system to be able to anticipate new threats and adapt to them. Beyond that, [3] also demands all links (communication, critical electronics and infrastructure) between aircraft and ground systems to be resilient to both failure and cyber-threats.

Of the surveyed guidance bodies, only ACARE suggests a timeline for these items to be developed. In [3] it suggests that by

2020 cyber security alerting mechanisms are in place,

2035 Commercial aviation systems can adapt to and overcome vulnerabilities, and

2050 It is possible to defend itself in-depth against potential cyber attacks.

In addition, Aeronautical Radio, Incorporated (ARINC) has published several standards on cyber security. However, while some standards by this organisation are accepted as a possible means of compliance by airworthiness authorities, they are not an airworthiness regulation. Moreover, currently they focus on security concepts (ARINC 811), data link security (ARINC 823), digital signatures (ARINC 835) and certificates (ARINC 842-1), as well as logging (ARINC 852). Consequently, they do not address aspects related to network security or indeed govern how to react to such incidents.

While there is no current regulation that gives quantitative guidance for cyber-security, several draft regulations and letters of intent have been published by airworthiness authorities. While these drafts are neither policy nor guidance, they can still provide a good indication of the direction regulatory efforts will take in the upcoming years.

International Civil Aviation Organisation

The United Nation's body for civil aviation, International Civil Aviation Organization (ICAO) 's published treatise on security [79] contains two paragraphs on cyber security.

Those two paragraphs mandate member states to protect critical information and communication technology from unlawful interference through cyber-threats. Furthermore, it is recommended to ensure the

- Confidentiality
- Integrity
- Availability

of critical systems and data through means of e.g.

- Security by design
- Supply chain security
- Network separation
- Protection from/limitation of remote access

However, no quantitative measures are provided.

European Union

The European Aviation Safety Agency is, according to its own website's FAQ section, not responsible for questions related to civil aviation security [50].

While a study was launched in 2014 to address the cyber-security needs of the Single European Sky ATM Research (SESAR), it has not been published (despite being promised for early 2015) [159].

In addition, the technical standardisation body European Organisation for Civil Aviation Equipment (EUROCAE) has prepared documents (ED-202A and ED-203) that are considered to be guidance material for certification within the EU. However, they only provide high-level requirements and describe procedural methods to include security in the airworthiness design process.

United States of America

While the airworthiness authority of the United States of America, the Federal Aviation Administration (FAA) has taken some efforts in the field cyber-security, there are no generally applicable rules for all aircraft. The most recent strategic plan (valid until 2022) [52] aims for the development of cyber threat models for critical aeronautical infrastructure. If an aircraft system is connected to the internet, individual regulation is applied for that aircraft type [43]. However, the US Government Accountability Office has repeatedly stated that further efforts need to be undertaken by the FAA to address cyber security. In its most recent report dated October 2020 it states several key areas for improvement [174], amongst them:

- Increasing cyber security risks to avionics system could impact flight safety in more connected airplanes,
- Implement oversight of industry mitigation of avionics cyber security risk,
- Focus on coordination on avionics cyber security risks,
- Improvement of the certification procedures of cyber security testing.

While the FAA is stated to concur with most of the positions in this paper, it will likely take several years until these issues are addressed and specific aeronautical cyber security regulation is published.

2.2.2 Automotive

Most of the requirements for aircraft detailed above also apply to the automotive sector. Notwithstanding, the standards differ. For functional safety, the International Organization for Standardization (ISO) 26262 standard [85] is used. It provides the classification of potential hazards based on three variables:

1. Severity,
2. Exposure, and
3. Controllability.

Here, *severity* measures the potential impact on human health of an event, while *exposure* describes the likelihood. Finally, *controllability* describes whether the impact can be avoided by timely reaction of a human (generally the driver of the vehicle). For each of those variables, discrete sets exist for classification. Subsequently, a measure of required risk reduction followed by a safety goal is assigned to each potential hazard. These safety goals are then used to derive functional safety requirements that specify how the effects of a risk can be mitigated [60].

However, here cyber security is only included in an annex that is marked 'informative'. It provides guidance on integrating cyber security aspects to prevent an adverse impact on functional safety. Yet in this document, only possible interactions with cyber security in the various stages of the product development cycle are listed. Neither quantitative nor qualitative requirements are given. As the authors of [115] point out, there is a significant gap in the maturity of standards in the domain between safety and security.

On an international level, the United Nations Economic Commission for Europe (UNECE) has recently published regulations [180] on the approval of automotive cyber security and its management systems. These will become mandatory in Europe for new vehicle types starting July 2022 and for all new vehicles produced after July 2024 [148]. In addition, the Republic of Korea and Japan plan an adaptation [173]. Specifically, this regulation mandates that manufacturers provide documentation related to

- Cyber security-related information throughout the supply chain;
- Risk assessment, test results and mitigations;
- Appropriate cyber security measures of the vehicle's design;
- Detection and response to cyber security attacks;
- Logging of data related to detection and forensic capabilities of attempted or successful cyber-attacks.

In addition, it requires that manufacturers to demonstrate a cyber security management system for development, production and post-production phases. Beyond that, it provides appendices listing high-level of vulnerabilities, attack methods and possible mitigations to be considered by the manufacturers. Furthermore, the ISO/SAE 21434 [82] has been developed as an international standard for road vehicles. It establishes minimum criteria for automotive cybersecurity engineering. Comparable to the UNECE regulation, it considers the entire vehicle lifecycle. However, it does not specify any preferred technologies, solutions or mitigation methods. In addition, autonomous vehicles and infrastructure requirements are not considered.

2.2.3 Smart Grid

In addition to stability and availability requirements, the smart grid is defined by the changing nature of demands (e.g. summer/winter) and supply, especially for renewable energy sources (sunny/windy/overcast). Consequently, in addition to the obvious privacy and security implications of smart meters that allow fine-grained energy consumption measurements, another security aspect that needs to be considered is the impact of demand response protocols. Those allow a bi-directional communication between consumers and suppliers to reduce the former's consumption when demand is high [141]. Consequently, beyond ensuring privacy of smart meters, it also needs to be guaranteed that those requests are authentic and that their integrity can be verified. Furthermore, no private information may be explicitly or implicitly shared.

2.2.4 Accountability

Accountability is a means to ensure both availability and reliability in safety-critical systems. By continuously recording the system state, possible problems can be identified and rectified in a timely manner. This can either be done after an incident or on a regular basis during maintenance. In some critical systems such as aircraft, the recording of several hours of past flight data is required by regulations [48]. In addition, records of all maintenance actions need to be kept for the entire lifetime of the aircraft [49].

Currently, such records are kept in paper form. However, considering the lifetime of several decades, it becomes evident that a paper-based record is problematic. Environmental factors (water, fire etc.) can damage or destroy the records, and tampering cannot be prevented.

To address some of the challenges detailed above, [99] proposes a so-called *security box*, which would be the equivalent of a digital flight data recorder (DFDR) for security breaches. To counteract privacy concerns, it is suggested to have it activated by crew when system does something unexpected (the authors called it a 'panic button'). IFALPA representing airline pilots [81] also recommends mandatory reporting of occurrences and malfunctions that could hint to a cyber-attack, either on the aircraft or on aviation infrastructure. Equally, the advisory council for aviation research in Europe [3] also suggest that incident data of security events is captured and made widely available to all stakeholders, as it is done currently with safety data.

Distributed ledger technology (DLT) could provide an alternative way of record keeping. This technology provides means to manage a digital ledger on a peer-to-peer basis and thus without a central authority. In principle, a distributed ledger is a decentralised database with a global state. It has gained popularity after it has been proposed as a basis for an electronic cash system in [129]. While the original purpose of a ledger is to track financial transactions, [130], any type of data record may be contained therein. Data records are sequentially appended so that they are tamper-proof, i.e. cannot be deleted or forged. The ledger is jointly maintained by several peers that need to agree on the current state, without trusted third parties or a central authority. Consequently, a limited number of malicious peers cannot manipulate the ledger's state. Consequently, it has been employed in a plethora of works, also for more general applications such as smart contracts.

Here, each peer keeps an identical, private copy of the ledger and broadcasts information to the others if necessary. Peers may initiate a *transaction* or change of the ledger's state, however the validity of the transaction is checked before the ledger is modified. A *consensus algorithm* determines the order in which the transactions are appended to the ledger. The implementation of these algorithms is system dependent. While alternative methods exist and are commonly used for financial transactions, the most interesting model for consensus in critical systems is based on Byzantine fault tolerance (BFT). This allows a system to operate correctly even in the presence of errors, irrespective of whether they have been caused intentionally or not.

2.2.5 Network Calculus

Somewhat complementary to the aspects covered in this thesis, network calculus [106] is a mathematical method that allows to analyse the worst-case network performance. It permits to obtain deterministic bounds to network flows, a key criterion for real-time applications in critical systems. As it uses worst-case considerations, it generally yields overly pessimistic results.

While there are continuous efforts to make it more realistic e.g. [59, 33] and applying it to critical systems, there is a significant gap between the obtained worst-case performance and average, measured performance. Consequently, this method generally results in suboptimal network designs.

What is more, it does not provide a means to obtain the network configuration in the first place. A candidate configuration has to be obtained by other means and may then be verified using network calculus. In addition, it only analyses the network as a whole and does not solve the issue of mitigating the effects of a detected anomaly.

2.3 SDN

2.3.1 Concept

In traditional networking, the network switches handling the data are also responsible for routing and network management. Consequently, the network switches need to be quite sophisticated and also monitor traffic of neighbouring nodes to decide which switch they should forward any given packet to. As the network performance depends on the algorithms employed, the implementation is usually proprietary. What is more, interoperability between different vendors is severely limited. Both aspects result in high costs for network components. Additionally, new features cannot be easily added, since every single switch would have to be replaced. Beyond that, operators of critical infrastructure might face the challenge to source replacement parts because the implemented technology might already be obsolete.

In contrast, one of the key features of SDN is the separation of the control plane responsible for network management and the data plane that is handling the actual data. This is done by introducing a new component, the *controller*, that has a global overview of the network. In traditional networks, the control of the network and traffic flow control within is handled by the network switches in a decentralised manner. In contrast, SDN switches are simple affairs that merely consist of input and output ports and *forwarding table entries* that tell a given switch the next destination of a packet that arrives from a given port, with a given source and a given destination. Alternatively, a packet can also be dropped, or if no rule is defined, the switch may request a strategy from the controller to deal with a given packet.

If the network changes, e.g. because a link is congested or additional traffic arrives, the controller is notified by the switches (either the affected switch or neighbours) and can modify the network by changing the *forwarding tables* in the switches. The communication between controller and switches can be either performed *in-band* i.e. using the same network as the data, or *out-of-band* using a dedicated control network. The comparison between traditional networking and SDN is illustrated in figure 2.1.

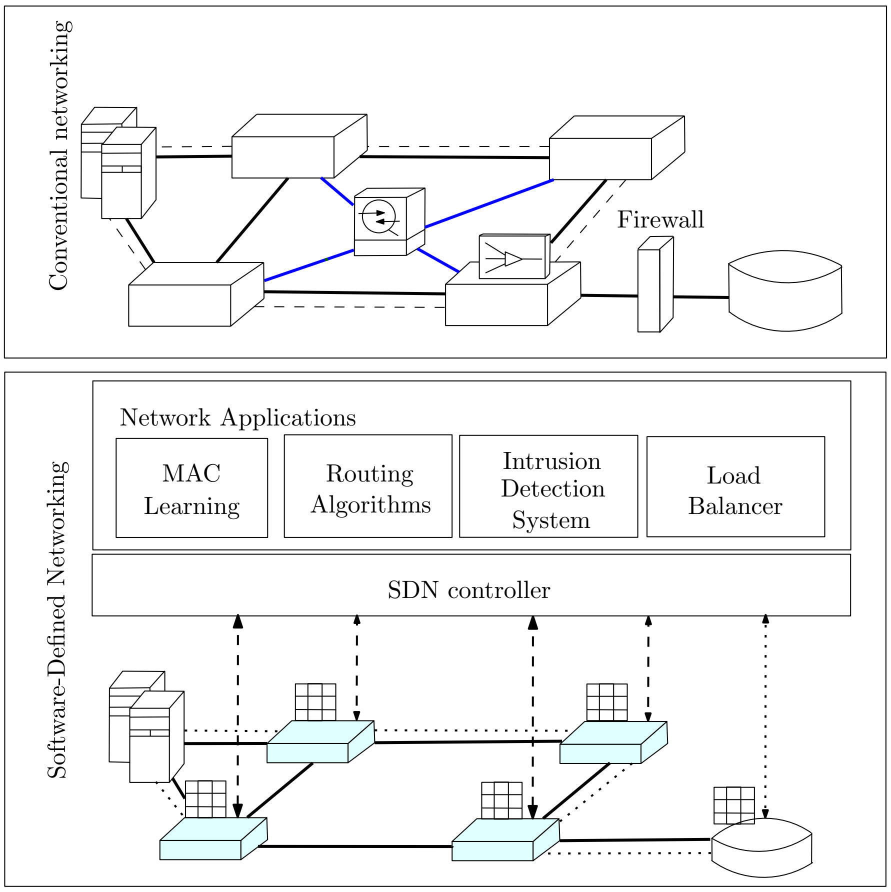


Figure 2.1: Comparison between conventional networking and SDN, amended from [103]

2.3.2 Functionalities

According to [103], the functionalities of SDN can also be logically separated in layers or planes. As the lowest plane, the *data plane* is responsible for the transmission of data packets, fragmentation of a data stream, forwarding and replications of messages with more than one recipient (multicast).

In the layer above, the *control plane*, routing tables, packet handling policies and service availability announcements are contained.

Depending on the size and requirements of a network, another optional layer may lie beyond the control plane: the *management plane*. Here, additional functionalities with respect to availability and control may be summarised. Examples of such include the treatment of new devices, reconfigurations in failure cases and activating/deactivating devices.

Since this centralisation would introduce a single point of failure (the controller), it is also possible to have more than one controller. Various strategies exist to coordinate between different controllers, e.g. dividing into several subnetworks. A more detailed overview of those will be provided below and in section 5.3.

2.3.3 Communication Between Switches and Controllers

As described above, the *control plane* provides means for information exchange between switches and controllers. One protocol for this communication is the *OpenFlow* protocol. It is vendor-independent and developed by the non-profit Open Networking Foundation.

In [133], three types of messages between the controller and switch are defined: *controller-to-switch*, *asynchronous* and *symmetric*. While the first type is initiated by the controller, *asynchronous* communication is used by the switch to update the controller about network events and changes to the switch state. Finally, symmetric messages may be initiated by either. Further information on the various communication types can be found in chapter A as well as in [133].

2.3.4 SDN in Critical Systems

To use SDN in critical systems, providing resilience is a critical issue. Here, both the control plane and the data plane need to be addressed separately.

Data Plane

Resilience of the data plane has been studied in various publications. For example, [161] presents a framework for SDN to abstract resilience functions through so-called *management patterns* to describe the interactions between different resilience mechanisms. These patterns specify requirements which are satisfied through the assignment to particular components by a combination of a knowledge-based approach and machine-learning based approaches.

In an alternative approach, [139] investigates the recovery from failure in SDN by performing run-time optimisation using iterative routing of feasible solutions until optimality is achieved. However, their only constraint is the link capacity, this they do not take the requirements of the traffic into account. A similar strategy is presented in [13]. While not using an iterative approach, it focuses on minimising operational cost when finding a recovery path while simultaneously trying to minimise the flow operations needed.

Other publications also investigated resilient routing for other types of networks. The change in complexity between resilient and non-resilient IP routing has been addressed in [70]. The authors compared various heuristic objective functions for their effect on link utilisation and average path length. It was found that while heuristics improve certain characteristics, others will be negatively affected. Thus special care needs to be taken during network design to select the appropriate strategy.

Optimisation for non-resilient networks has also been a prospering research topic. Among them, [147] focuses on minimising path length of packet forwarding and switch memory usage under the constraints of forwarding table entries. On the other hand, [137] extends the problem by also considering the facility placement i.e. where the traffic sources and demands are placed. While their algorithm dynamically changes traffic routing and demands, it does not consider delay-sensitive traffic.

Non-functional safety requirements have been less frequently considered. For example, [76] investigates a delay-constrained routing problem for a M/M/1 queue, where job service rates follow an exponential and arrivals a Poisson distribution. In contrast, the authors of [28] minimise the delay of the flow with the highest delay bound in the network using shortest path and greedy algorithms as well as iterative versions thereof.

Beyond that, some papers have also investigated the safety-critical use of SDN. While [153] reviews general challenges and security issues, [116] provides an overview of how SDN (positively and negatively) can influence the network resilience. Finally, [146] describes a mechanism to provide one-link fault tolerance by using the fast-failover groups feature of OpenFlow. In addition, while not technically addressing safety-critical systems, [72] studied SDN for deterministic networks using an in-band configuration channel.

Control Plane

While the data continues to be forwarded by switches after a controller failure, new flows or failures in the data plane cannot be dealt with. Consequently, using several controllers to provide resilience against controller failures is a commonly used method. However, several strategies exist to coordinate between those controllers, which will be discussed in the following paragraphs.

In [152], the authors address both intentional (i.e. attacks) and unintentional failures of the controller. Either one primary controller handles all the traffic and other controllers stand by in case an anomaly is detected, or *packet-in* messages are used to forward client requests to primary and secondary controllers. Either way, the switch then collects the controller messages and evaluates them for inconsistencies. To this end, the authors add components to the network to reassign controller-switch connections, to compare controller messages at the switch, and to evaluate controller-to-controller state synchronisation messages. They use replication of control plane computation and transmission to target switches, which then need to evaluate these messages for correctness. To avoid inconsistencies in the processing of requests of the same resource, controller responses are delayed until the required number of control messages has been generated. Finally, an integer linear programming (ILP) problem is formulated and solved after each controller failure detection. The constraints considered were bandwidth and delay limits.

In contrast, [51] proposes a stepped remediation mechanism and multi-level monitoring and integrate a production-grade distributed controller, ONOS. It essentially consists of a collection layer where alerts from IDS systems such as Bro or Snort, OpenFlow (OF) resource monitoring as well as standard network monitoring components such as sFlow are gathered. They are then processed in a coordination layer that interfaces with an application layer where additional security applications are hosted. The distributed controller instances are clustered and then share data amongst themselves. Consensus within each cluster is reached by using the Raft algorithm. Additionally, the monitoring level of sample-based monitoring can be dynamically adapted. To benchmark controller performance, *cbench* was used.

The concept of *self-stabilisation* to deal with several concurrent failures is introduced in [26]. The authors use in-band management to recover from controller, switch, and link failures. However, malicious controllers and the necessary countermeasures have not been considered. Beyond that, the effects of using the Raft consensus algorithm to coordinate the activities of several controllers on system response time is investigated in [151]. The authors use stochastic activity networks (SANs) for modelling and numerical evaluation of distributed SDN clusters. They assume a mechanism that monitors the health and correctness of controllers.

In [176], the authors propose to cache flow rules and distribute them across multiple controllers in order to mitigate controller saturation and rule modification, thus providing Byzantine fault tolerance and reduce flow setup delay. The issue of control plane inconsistencies and their effect on network performance is highlighted in [21]. To address this, the authors propose a distributed SDN control plane architecture. The main feature of their proposed controller design is its strong consistency due to a replicated data store to ensure a consistent network view. However, this method only addresses unintentional faults, not attacks.

Some papers also consider the effect of using several controllers in critical systems. For example, [156] discusses the use of SDN for heterogeneous and time-sensitive industrial internet of things (IoT). The focus lies on the communication between distributed SDN controllers with legacy data planes. They present a cascaded controller for time-sensitive network (TSN) using legacy hardware. The controller is integrated into a PLC. While the proposed architecture includes several SDN controllers, only communications between controllers governing different hierarchies of industrial applications are considered. Resilient controllers are not investigated.

Another essential aspect for critical systems is the integration of legacy devices and systems. This has been detailed in [65]. That work addresses the issue of legacy devices in SDN with a focus on synchronising the controller and network management system (NMS) for cases of bring your own device (BYOD). NMSs are used for configuration, management, and troubleshooting in order to maintain and supervise networks. For the synchronisation of legacy and SDN control, polling, pushing, and direct synchronisation were investigated. A detailed overview of the security challenges associated with SDN can be found e.g. in [157].

Using dynamic network reconfiguration in critical networks also requires traceability of the implemented changes. This is to ensure that any potential issues are detected and rectified as soon as possible. Amongst the few works that address this essential issue, [1] investigated the use of ledgers for SDN. However, their application was privacy-preserving identity management.

In [61], the impact on the performance of the underlying physical system of SDN controller failures has been studied. They identified the following failure cases:

- Random failure
- Delayed response by the controller

- Error control messages (rule modification, deletion, and message drop)
- Infinite loops
- Resource exhaustion

Yet only the impact of those cases has been studied, and not possible mitigations.

2.4 Addressing Incidents in Safety-Critical Systems

As discussed in section 2.2, safety-critical systems require mechanisms to address incidents. On a network level, SDN 's ability to reconfigure the network during runtime can be a means to facilitate this. This section will provide more details on how such incidents can be addressed using SDN as well as other means available in safety-critical systems. While the focus will lie on means to detect security events (i.e. intentional faults and failures), detecting and mitigating safety-related events (i.e. unintentional faults and failures) is equally important for safety-critical systems. Moreover, usually only the *effect* of a fault or failure can be observed, thus it is not always possible to clearly differentiate between the two. Consequently, this thesis will address both intentional and unintentional faults and failures.

In one of the earliest works to consider reactions to incidents in safety-critical systems, [171], the authors propose a framework for a Smart Grid SCADA system composed of the following four key components:

- Real-time monitoring,
- Anomaly detection,
- Impact analysis, and
- Mitigation strategies.

While the authors specifically wanted to address security incidents, these components are also required to cover safety incidents.

The first component that is required is a real-time monitoring capability. On the one hand, any incidents need to be detected as soon as possible to limit the impact on the system. On the other hand, it needs to be ensured that any monitoring does not delay real-time, safety critical traffic.

Secondly, anomalies need to be detected in the monitored traffic. Here, the sensitivity of the algorithm is crucial. A high sensitivity will result in a high false-positive rate, which significantly increases resource usage. In contrast, low sensitivity means that incidents go undetected.

Thirdly, it is necessary to establish the impact of the detected anomaly. Here, a variety of parameters such as delay, resource usage etc. can be used. The detected impact can also help to decide whether a reaction is needed at all or whether the anomaly detection has produced a false positive.

Finally, the impact warrants a reaction, mitigation strategies are necessary to address the detected anomaly.

In the following pages, a detailed overview of research efforts into those areas (monitoring, anomaly detection, impact analysis, and mitigation) will be given.

2.4.1 Real-Time Monitoring

As detailed above, through the increased interconnections with cyber systems, critical systems are being more and more exposed to cyber risks. Consequently, in order to guarantee safety-critical functionality, these systems must also be monitored and protected by the same means as traditional cyber systems.

For monitoring, two approaches exist: either using dedicated equipment for anomaly/intrusion detection, or integrating monitoring functions into functional code. While the former is commonly found in traditional computing infrastructure such as servers, integrating monitoring functions is being used in safety-critical systems to detect and contain faults early. Notwithstanding, it may also be used to identify attacks, e.g. by monitoring privilege escalation.

Network Anomaly Detection/Intrusion Detection

Intrusion detection systems dynamically analyse network traffic as well as artefacts such as log files to identify changes that could point to an attack. Generally, two distinct approaches exist: either identifying known attacks (*signature-based*) or detecting changes in traffic behaviour (*anomaly-based*). The algorithms used can vary in complexity, from simple white- or blacklists and firewalls to complex systems that use machine learning to dynamically react to network changes. Some techniques such as frequency analyses also work on encrypted traffic [102]. Recent survey papers about the techniques used include [97, 47, 124].

Irrespective of their complexity, output from such systems can either be used directly for mitigation (e.g. by ensuring that traffic from a compromised device is dropped at the next switch), or for more complex analyses to detect previously unknown vulnerability and react accordingly e.g. by patching specific code.

Integrated Health Monitoring

Another component available in most safety-critical systems that can also be used to detect anomalies is integrated health monitoring. In critical systems, checking whether relevant parameters fall into specific boundaries is checked as part of functional code execution. Consequently, if a parameter falls outside the specified operational condition, no exception is raised, but rather the code automatically addresses the fault. In the simplest case, the fault is simply logged, but more complex reactions such as triggering a maintenance action if a fault recurs are also possible.

In aviation (where it is termed integrated vehicle health management (IVHM)), most software components include checks that verify both input parameters of sensors, and also the correct execution of code within a software function.

In addition, these small pieces of software integrated into functional components can detect buffer overflows etc. While they are currently only used to detect the necessity for maintenance and sensor malfunction, they can also be used as an input to detect faults and anomalies, e.g. a switch malfunctioning due to overheating.

For aircraft, the ARINC 653 standard [7] gives details as to what information can typically be included in health monitoring:

- Watchdog timer
- Checksums
- Memory protection
- Timer interrupts
- Privilege execution violation
- Buffer overflows
- Reasonable output
- Measurement of execution time
- Deadline misses
- Operating system (OS) monitoring functions

While this standard focusses on the detection of natural system degradation due to ageing components, some of theses provisions (detailed below) can also be used to detect cyber incidents.

Watchdog timers are used to regularly measure a function's execution time. This could be indicative of a variety of conditions, both benign and malicious. For example, it could indicate that function is waiting for a semaphore that is not freed, or that code is executed that is not supposed to be executed. An equally important variation are *deadline misses*, where a function has not finished or information has not been transmitted at a given time. This is especially relevant for real-time systems, where a given information looses its validity after a certain time. While such outdated information is not relevant for further program execution, it is relevant to measure the occurrence of such events, as only a certain percentage of such information can be dropped without being indicative of problems. Slightly different, the *measurement of execution time* is used to detect continuous variations of execution time, while a watchdog or deadline miss would only detect a hard violation. Along similar lines, *timer interrupts* stop the execution of a program after a certain time.

Furthermore, *checksums* can be used to verify the integrity of transmitted messages. To this end, an algorithm is applied to the message and the result appended prior to transmission. The receiver then applies the same algorithm to the message and checks whether the checksums match. This method can be used to detect bit-flips (most commonly occurring due to solar radiation) or intentional tampering. It should be noted that checksums increase the traffic flow, as they increase the length of the original message. Additionally, the calculation of checksums requires time and computational resources, consequently they can only be used for a certain proportion of messages.

Another group of monitoring functions relate to memory usage. *Memory protection*, on the one hand, prevents a function from writing outside of its allocated memory, which could be indicative of an attack or a code malfunction. In a similar field, *buffer overflows* occur when a function tries to write more data into an allocated buffer e.g. a 64 Bit word into a 32 Bit buffer. While some statically typed programming languages such as Ada [84] are designed to prevent this automatically, it is important to monitor as it could be indicative of a code malfunction. Additionally, such buffer overflows can be intentionally triggered, thus leaving a system vulnerable if they have not been addressed properly.

Furthermore, it is necessary to monitor whether code tries to run outside its assigned privilege level. Such *privilege execution violations* could indicate an attack e.g. when a low-priority management function tries to execute code with the highest level normally associated with safety-critical functions.

Finally, monitoring whether *outputs are reasonable* is another important feature. For example, a temperature sensor on Earth cannot encounter temperatures below $-80°C$, hence if such an input is detected, it can be considered to be an anomaly and the sensor is likely malfunctioning.

2.4.2 Anomaly Detection

Anomaly detection can be used in a variety of applications, from tracking down software bugs, vessel movements, social media analysis to medicine and network security scans [69, 122, 111, 46, 19, 94]. Moreover, anomaly detection can also be and is frequently used to detect abnormal situations in safety-critical systems, such as nuclear power plants, across an airline's fleet, in a power network etc. [see: 23, 41, 170]. However, the decision on the correct reaction to a given anomaly usually remains with a (human) operator. For security purposes, everything that deviates from allowable, "normal" behaviour is considered to be an intrusion. A sample of normal system behaviour is usually obtained by recording the system/network for some time.

Principles

Generally, the following sequence is performed by anomaly detection systems [see 87]:

1. Data acquisition
2. Data partitioning
3. Probabilistic finite state machine construction
4. State probability generation (recursively computed to approximate the dynamic system)
5. Class information identification

To begin with, data needs to be obtained from the system. This process has already been described in the scope of monitoring as detailed above. Subsequently, the data needs to be partitioned, as not all data can be inspected in real-time, if the processing speed of the anomaly detection is slower than the transmission speed of the network. Based on the data thus obtained, a finite state machine is constructed. Depending on the data collected during system operation, the anomaly detection system estimates in which of the states the monitored system is at a given time. Subsequently, the classification information is identified.

The anomaly collection logic itself can be grouped into three basic types [154]:

Heuristic This is the most common type, which may also use machine learning methods. Subsequently, a statistical analysis is performed to obtain a profile of the system. However, malicious behaviours that are already present during the learning phases cannot be detected.

Policy-based (aka. knowledge profile) This type relies on a human expert to build a profile of the environment based on known data. This is only manageable for small and constrained environments.

Mixture of heuristic and policy-based This type depends on structured learning, thus allowing to obtain a profile more quickly and with higher accuracy.

Modelling and Classification Algorithms

To distinguish between normal and abnormal behaviour, three different basic methods can be used: statistical methods, analytical methods, and machine learning. The two former methods are generally faster and more deterministic, but the resulting accuracy depends on the definition of thresholds. Moreover, it is challenging to obtain the data set used to obtain profiles of normal/acceptable behaviour. On the one hand, legitimate and unmalicious behaviour will frequently deviate from normal profile, leading to a high false positive rate [138, 42]. On the other hand, any undesired behaviour that is present in the original data set will not be considered to be an anomaly during run-time. Consequently, practical applications benefit from not only basing decisions/classifications on anomaly arbitration because this causes a high rate of false positives but also on other factors such as network status. It should be noted that any changes in the monitored system also require a retraining/adaptation of the anomaly detection model. Moreover, some models are susceptible to *overfitting*, where the model it too tailored to the learning set, so it does not perform well on test data or the real system. Modelling and classification algorithms for anomaly detection include [164, 154, 2, 100]:

Artificial neural networks are one of the earliest machine learning mechanisms and have been theoretically discussed in the 1960's but due to technological limitations, the first implementations were later [109] and eventually also used for pattern recognition [20]. They act as an computational imitation of biological/neurological processes, consisting of a certain number of layers between the input and output layer (which are visible to the user). They act as function approximators and consist from a sequence of decision units. While they can learn from continuous data, multiple outputs and decision boundaries at once, but are very inefficient when surplus data is included in the training set and can suffer from overfitting because they can create arbitrarily complex models. Works that used it for anomaly detection include [136, 158, 118] and for fault detection [131, 158, 128, 121, 118].

Bayesian models are statistical models that have several subtypes, depending on the algorithm used. In general, they only require a small number of training samples and can learn uncertain concepts without overfitting, but assumes that data can be represented by single probability distribution. Several subtypes: *Statistics/interference*: the most basic model using only statistical methods. Has been used e.g. in [71] to obtain a patter for interaction with a network and in [37] to detect malicious users. *Naïve*: tends to ignore inherent time dependencies in the behaviour, recent publications include [182] for anomaly detection (AD) for industrial applications, or [186] as one of several methods used comparatively. *Analysis*: here, a probability is assigned to a hypothesis. *B. linear models*: used in [165] to analyse reliability (together with a support vector machine, see below). *B. networks*: for estimating the system health of industrial machines [25] to determine when maintenance is necessary or used when monitoring of maritime vessels [122] or, more recently, for resilience analysis [185].

Decision trees predict data labels by iterating the data through a learning tree. This method only works with linearly separable (i.e. non-continuous) data. Like neural network, they suffer from overfitting due to their ability to generate arbitrarily complex models. Moreover, they do not work well with data that requires diagonal partitioning. Furthermore, all conditional statements are sequenced. For example, they have been used for the identification of unknown threats [44], in wireless IoT networks [10], and in [9] to detect anomalies in SDN.

Fuzzy reasoning can not only accept binary classifications, but also values that lie in between. Consequently, it can be used to represent more finely-tuned anomaly classifications (e.g. giving a probability that something is an anomaly). Recent publications include [68, 175, 86].

***k*-means clustering** clusters data around k centres that represent the mean of those values. The number of centres has to be defined by the user. This algorithm groups data objects so that members of the same group are similar to each other, while members of different groups are dissimilar [11]. This measure of similarity (also called distance) can then be used to establish quality measures for clustering. k-means does not need high computational power and is simple to implement, however it is very sensitive to irrelevant data and the complexity is linear. One of the earliest works that use this method to detect malicious activities in their university network is [120], with more recent efforts applied on flow records [127], together with decision trees in [58], and [32] for industrial control systems.

Markov models are used in systems with state changes. There are several subtypes of Markov models, depending on the system characteristics (discrete or continuous), state space and time. To obtain an accurate Markov model, it is necessary that the anomaly detection occurs before the data-tagging, otherwise relationships between anomalies may not be discovered. This unsupervised clustering method has a commonly-used variant: *Hidden Markov Models (HMM)*: method for behaviour modelling that uses a structure of nodes with transition links to represent the time aspect, commonly uses the Baum-Welch algorithm [17] for training and the Viterbi algorithm [56] to find the optimal model [31], but tends to suffer from overfitting; [189] use a HMM to distinguish faults from attacks, *Markov chains*: in one of the earliest examples, [183] model temporal behaviour of anomalies, [140] estimate the probability probability from past anomaly-free observations, while it is applied by [78] to obtain an adaptive model of SDN intrusions.

***N*-Gram features** work by selecting byte sequences of length n that are present in the data and represent structural components and fragments of instruction and data. While it has initially been used for text categorisation [29], the method soon received attention for anomaly detection. An detailed description of $n - gram$ models used in intrusion detection is provided in [181]. Recent works that used this method include [96] (in combination with a support vector machine), [112] (for anomaly detection of flight data) and [45] (to detect privilege violations, in combination with support vector machines).

Principal component analysis (PCA) allows to compress data and present it as a new set of variables (termed *principal components*), ordered by the amount of information content [2]. This allows to find common features in labeled data sets (i.e. where it is known that something constitutes an anomaly). These features can then be used for anomaly detection in online applications. While the mathematical principles have been developed in the first half of the 20th century, its use for analysis of computer networks came with [105], while more recent publications using a variety of this technique include [68, 40, 54, 24]

Support Vector Machines (SVMs) classify data using labeled training samples, work well on multi-dimensional problems and continuous features, but require a significant amount of time to complete. [149, 187] employed this technique for fault diagnosis, while [165] used it to investigate anomalies in data from an aircraft manufacturer's server and [135] to analyse aeronautical system health. With some adaptations, it has also been used to learn from partially labeled data sets, e.g. [177].

Further details on those and further methods can be e.g. obtained from [19].

Anomaly Detection Using SDN

In addition to the generic methods detailed above that work on many different types of data, SDN itself also provides means to detect network anomalies. As reconfiguration based on variations in network demands are a key feature of SDN, standards such as OpenFlow natively include means to monitor the network state.

To allow the controller(s) to have a current picture of the network state, the following information is provided by the switches, either periodically or upon request by the controller [133]:

- Link up/down
- Switch up/down
- Keep alive message exchanges
- Statistics such as number of flows, queue size, bandwidth etc.

First of all, switches provide information to the controller if they detect that a link connected to them is up or down. The same applies if a switch is down, while a *switch up* event message is sent directly from the activated switch to the controller [133].

In addition, the controller can poll all switches regularly to send alive messages, and provide statistics about number of flows that are being handled, queue size, bandwidth used etc. Obviously, the polling frequency has an impact on network load, especially if the communication between switches and controller is in-band and not on a separate control network.

Challenges to Anomaly Detection

One of the main challenges associated with anomaly detection is that it is rather difficult to obtain profiles of normal/acceptable behaviour during system runtime. The main problem is to ensure that no intrusion/malware is present during that initial phase, since these events will not be considered to be anomalies in the future. If they are present, the running system/network will be vulnerable to these types of attacks, as they are not considered to be abnormal.

What is more, even legitimate and non-malicious behaviour will frequently deviate from the normal profile, leading to a high rate of false-positives [138, 42]. Thus, some authors suggest to base decisions/classifications not only on anomaly arbitration to reduce the rate of false positives [34].

One method to reduce this high rate of false positives is to consider human behaviour as part of the anomaly detection system [see 107], for example the difference in traffic volume between day and night etc. However, this approach is only valid for safety-critical systems where there is significant interaction with human beings such as the smart grid. Notwithstanding, even traffic networks that are exclusively used for machine-to-machine communication can vary periodically (e.g. depending on energy used in processes in manufacturing plants, phases of flight in airplanes etc). Consequently, the false positive rate may also be reduced in those application by taking periodicity into account.

Another issue is that the more network traffic there is, the easier it is for malicious traffic to hide in plain sight. Heuristic malware detection programs are usually either static or dynamic, where static heuristics usually employ indicators such as structural anomalies or program disassembly, and dynamic heuristics use runtime indicators frequently obtained from virtual environments, also called *sandboxing*. Both attempt to detect zero-day attacks (i.e. to detect new malware at launch, and not only after thousands of machines have become infected), but since both approaches have shortcomings, they are usually combined [44]. Other approaches use more complex mathematical models, e.g. [110] use an analysis based on game theory to predict attacks.

Known Avoidance Strategies for intrusion detection (ID) Methods

Beyond this, with more sophisticated detection mechanisms, attacking techniques have also become more sophisticated. Consequently, they have been able to circumvent protections by IDSs.

Five basic penetration techniques have been demonstrated by literature to be able to evade queries by an IDS [36]. Firstly, *packet splitting* (including IP fragmentation and TCP segmentation) separates IP datagrams/IP stream into non-overlapping entities, thus bypassing the IDS, unless the (usually) small entities are reassembled by the IDS. Secondly, denial of service (DOS) attacks may be able to slow down the rule-matching algorithm in Snort significantly by modifying the input traffic, also know as *algorithmic complexity attack* [163]. Thirdly, in *duplicate insertion* attacks, overlapping or duplicate entities are used to cause inconsistencies within the IDS, if information about network topology or the victim's operating system is missing. Fourthly, *payload mutations* allow an attacker to transform payload that the IDS would classify as malicious into equivalent - but not detectable to the IDS - payload. Finally, *shellcode mutation* encodes, encrypts or compresses shellcode to mask its signature, thus bypassing an IDS that detects shellcode.

Some of those techniques have also been implemented as tools which facilitate and automate such attacks. *Fragroute* and *Sploit* are able to perform packet splitting and duplicate insertion, the latter is furthermore capable to mutate payload and shellcode. Other tools for payload mutation are *Nikto* and *Havij*, while *ADMmutate* and *Metasploit* can manipulate shellcode. Further details on these tools can e.g. be obtained from [36].

Beyond that, obfuscation is a technique for malware writers to obscure code so that is it functionally identical to an original code, thus making it difficult to analyse. Since a popular approach for anti-malware software is signature detection (i.e. comparing a file's content to a database of known malware signatures), this is a rather successful attack path. Further detail on obfuscation techniques can be found in [132]. However, obfuscation could be made a less effective tool if intrusion/malware detection algorithms were to shift away from signature detection and towards behaviour detection of malicious code.

Furthermore, dynamic analysis systems avoidance mechanisms are able to circumvent code sandboxing. In this context, two main methods exist. Firstly, *stalling code* performs unnecessary but harmless calculations until the sandbox times out. Subsequently, malicious code is executed on the target machine [98]. Secondly, sandboxes have *blind spots*. These occur as sandboxes add extra bits of code that send notifications for function or library calls, which can be detected by the malware. Moreover, between the calls of this code, the sandbox cannot see what the program is doing, which is hence exploited by the malware.

To conclude, while a plethora of methods exist to detect anomalies, attackers continue to try to bypass detection mechanisms. Consequently, relying on any one method, however good, will not result in complete system coverage.

2.4.3 Impact Analysis

Once an anomaly has been detected, the impact on the monitored system has to be established. In principle, the impact can be matched to the satisfaction of requirements. Consequently, the impact on

- Confidentiality,
- Integrity,
- Availability,
- Resilience against faults and failures, and
- Timing guarantees/predictability for real-time systems

can be measured with respect to their impact on safety and security.

Here, the impact of an individual anomaly on the entire safety-critical system has to be determined for the criteria above. While the precise weights of these aspects need to be determined individually for each safety-critical system, safety takes precedence over security. Consequently, detected losses of confidentiality might be found more tolerable than loss of guarantees for timing and predictability. It should also be noted that losses of resilience are unproblematic while at least one system/network path remains to serve the critical demand.

Based on the impact assessment, the system can determine whether an action is necessary or whether the detected anomaly is considered to be a false positive. In addition, even a detected anomaly might not require a reaction if it is transient or does not have a detectable impact on the system.

2.4.4 Mitigation Strategies

If the impact on the system has been deemed to be sufficiently great to warrant some adaptation to the system to mitigate the effects of the detected anomaly.

In [64], risk mitigation is defined as the risk associated with a given mitigation action in percent. It is calculated as the product between the percentage of the attack covered by the action and the percentage of reduction of the total incident cost through mitigation. However, this work focusses on identifying and protecting resources rather than traffic constraints. Moreover, the only focus is on attacks and not on failures that might also require responses.

Self-Protecting Systems

Self-protecting systems are able to autonomously fight back intrusions in real time. They usually merge several intrusion detection systems. Various intrusion detection methods can be used to detect the illegal behaviour of a device. The reaction to a detected intrusion depends on the application. It is either possible to isolate the misbehaving device and all the software running on it by removing all the connections with the network, or to force the failure of the incriminated device (fail-stop). Again, there is no general recommendation on which method to prefer, it depends entirely on the application [42].

One possible method for such a system is using the *principle of the least privilege*. Here, undeclared communication channels are trapped automatically and a recovery procedure is initiated. The authorisation of communication channels is automatically obtained from the system's software and hardware architecture and subsequently used to generate protection rules for the system [42]. In [93], an auto-regressive integrated moving average (ARIMA) model is used, which is efficient for short-term forecasting in a function of the parameters of p (previous observations), derivatives of historical data d and the correlation with previous errors (q).

Going even further, [169] propose a fully autonomic system, which in addition to being self-protecting is also self-configuring, self-healing and self-optimising. Generally speaking, all systems that are either self-optimising, self-healing, self-protective or self-configuring are described as autonomic [12]. These attempt to reduce the human influence on systems.

Another way to mitigate software faults, is *software fault isolation.* This technique modifies the client code so it can only write within a certain, designated area [38]. However, there are dynamic analysis system avoidance methods that can bypass this isolation technique. In addition, demonstrating safety with such a modification to running code is highly challenging but necessary for safety-critical systems.

Another approach is taken by [104]. There, the propose the use of a network emulator without the failed elements and then installing the difference ruleset between the emulated and real network.

In contrast, *hardware-based security applications* can be an alternative to classical, software based system defences. Using hardware-based security solutions is motivated by the fact that software-based applications only have a limited throughput. What is more, since security applications usually have a high computational cost, using hardware-based methods frees resources in the host processor and supports a parallel execution of operations. It goes without saying that hardware-based solutions are less flexible than security software, hence it is necessary that hardware is reconfigurable, a property of field-programmable gate arrays (FPGAs) [34].

Fault Tolerance and Resilience

Another way to address anomalies is to ensure that they do not have a negative impact on the system. To this end, fault tolerance and resilience are used. The former ensures that the system operates as intended despite the presence of faults. In contrast, the latter provides an alternative to the failed component. Here, it needs to be ensured that the failure does not affect all redundant part simultaneously, e.g. by using different manufacturers, different coding teams, different means of power supply etc. The following paragraphs will provide more detail on those concepts.

Initial studies into fault tolerance by von Neumann and Shannon in the 1950s [126] focussed on the telephone switching system. Subsequently, being able to continue to operate despite the presence of faults quickly became of interest for critical systems in aerospace in the mid-1960s with Jet Propulsion Laboratory's self-testing and repairing computer [15].

To prevent a random, individual fault from resulting in a system failure, *redundancy* is used. To support this, various techniques exist for both hardware [114] and software [150, 35] components, strategies which have been used since the 1970s. Notwithstanding, fault tolerance cannot address correlated failures, consequently it is necessary but not sufficient to provide resilience. Various authors have studied the dependability of networks. For example, [30] present a framework to study how dependable networks are and how they perform under perturbations.

In order to effectively prevent a single cause to affect all layers of redundancy and ensure surviveability, diversity in as many aspects as possible is required. While not trivial to quantify, [168] formalised it using set theory and state machines as follows:

$$\text{Survivability} = \left\{ \begin{matrix} S \\ E \\ D \\ V \\ T \\ P \end{matrix} \right\} \tag{2.1}$$

where S are acceptable service specifications, E potential and D actual system degradation due to external challenges, V describes the relative ordering between acceptable and actual service values, T valid transitions between service states under a given challenge and P the probability that services need to satisfy dependability requirements.

In addition, it needs to be considered that system need to remain operable despite an interruption to connectivity. This *disruption tolerance* is especially relevant in the context of an in-band SDN control channel, where connection to the controller might be temporarily unavailable if data plane communication takes priority.

Resilience is one of the key properties of a safety-critical system. Figure 2.2 gives an overview about the various sub-disciplines that affect resilience. Here, the dependability is further subdivided into aspects relating to reliability, maintainability, and safety, while security needs to consider aspects of confidentiality and non-repudiation (proof of the integrity and origin of data). Equally, quality of service, availability, and integrity need to be taken into account.

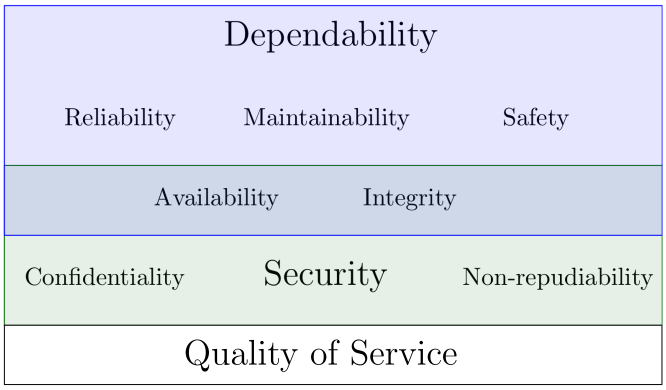


Figure 2.2: Resilience disciplines, amended from [168]

While not considering safety-critical networks in particular, [162] is one of the earliest works that considers resilience to faults and failures as well malicious attacks. Based on the model proposed in that work, this thesis considers resilience as the property of the network that allows to maintain an acceptable level of service in the presence of malicious attacks, human mistakes (e.g. misconfigurations), and software and hardware faults. While their original definition also includes large-scale natural disasters, this aspect will not be considered in this thesis, as only large and distributed critical systems such as the power grid needs to satisfy this criterion. Furthermore, [162] proposes a framework that describes a closed-loop control mechanism to allow networks to adapt and improve.

With respect to modelling and predicting dynamic system behaviour, various methods exist. For example, [185] utilise a Bayesian network. Another example, [155] and further developed in [161], a framework is proposed to manage resilience in networks. In addition, the authors argue the following aspects of resilience:

- malicious attacks,
- software and hardware faults,
- human mistakes (e.g. misconfigurations)
- large-scale natural disasters

In [39] the impact on resilience of a (not necessarily SDN) controller in a self-adaptive network is considered.

Resilience in SDN has also been a prospering research topic. Generally speaking, SDN can adapt to changes in its system, environment or goals. One of the first approaches into controlling software has been performed by IBM in its autonomic computing initiative [95] with a control layer know as monitor, analyse, plan, execute (MAPE). While not specifically related to SDN, [39] investigated the resilience of such systems, pointing out that this lack of resilience is one of the major inhibitors of commonly using self-adaptive systems. The authors of this paper focus on the impact of controller failures on critical middleware for power plants caused by malformed inputs from measurement probes. They also aim to establish the impact of controller failures on resilience and modelled as discrete-time Markov chains.

Using a method introduced in [13], [14] formulate integer programs employing combinations of minimum cost and minimum number of operations to investigate the trade-off between path cost and flow operations. They also propose algorithms with computational complexity similar to Dijkstra's algorithm. The authors also stipulate that it is safer to leave existing forwarding table entries and merely give back-up paths a higher priority. In this case, it was found that a lower improvement in operation cost is to be expected.

However, due to the centralisation of the controller, [55] propose to synchronise the network's state information among multiple controllers. In [178], *ResilientFlow* is proposed to address large-scale link failures in SDN. To this end, modules are deployed on every switch so that switches may maintain their control channel i.e. the logical connection to the controller. These modules are permitted to modify flow table entries to this end.

Rerouting Traffic

In addition, SDN provides the means to reroute traffic to react to network changes. Consequently, this capability can also be used to mitigate the effect of node and link failures by rerouting traffic so that the failed components are no longer used. This method requires that there is sufficient link capacity available to deal with the additional traffic on the remaining paths.

Heuristic Algorithms: A plethora of heuristic algorithms to dynamically reroute traffic during runtime have been presented in literature. However, as safety-critical systems generally require performance guarantees and often deterministic behaviour for certification, such algorithms are not suitable. Hence only algorithms and implementations with path computations performed prior to deployment will be discussed in the following paragraphs.

The authors of [117] devise an algorithm that calculates reliable shortest end-to-end paths. In this work, reliability is defined as "a link between any two source/destination nodes with more shared edges", consequently the path with the highest number of shared edges from all paths between source and destination is considered to be the most reliable as the recovery time would be increased otherwise.

Backup paths are computed in advance in [5] to ensure network connectivity in the presence of individual node or link failures. To this end, the adjacency matrix of the failed component is used to calculate shortest paths using Dijkstra's or the Bellman-Ford [18] algorithm. However, creating node and link disjoint paths are not the focus of this work. Yet to ensure that there is no single cause of failure, this is a key requirement. Another closely-related concept are *tuples*. Tuples are lists of elements and can be used to reduce the number of packet specification rules. If rules they match closely, the usage of tuples has a predictable search time [34].

In [66], network design in response to link failures is considered. This work aims to minimise the cost subject to capacity constraints using a stochastic approach. While this work aims to minimise the number of shared links between standard and backup paths, single causes of failure may still occur.

Policy-based configurations and interactions between various resilience mechanisms on a high level of abstraction are introduced in [161] by the name of *management patterns*. These patterns describe how mechanisms such as firewalls need to be reconfigured to mitigate the effects of a detected network anomaly. As such, they are defined in terms of *roles* that represent OpenFlow functionality, and are hence associated with management functionality. Such management patterns are created offline prior to deployment, and assigned to the respective application instances during runtime. The case study of this work was monitoring web traffic and dropping traffic originating from malicious hosts.

Yen's algorithm [184] provides a means to find k shortest paths. It has been used by [147] to obtain feasible solutions to provide alternate paths when needed. The most appropriate path for failure restoration is then selected based on a model of network flows by performing network-wide, multi-objective performance optimisation. Yet the authors aim to separate performance and correctness in this approach, while those two properties are intertwined in critical systems.

In contrast, [91] propose to maximise rule sharing between flows to different endpoints to limit ternary content-addressable memory (TCAM) usage. Subsequently, the rules are distributed across the network using a heuristic approach to facilitate changes in network policies. However, traffic requirements are not taken into account. Thus employing a heuristic algorithm may result in some constraints being violated. In addition, the capacity of the TCAM is only a significant problem for large networks or those with many distinct flows.

Furthermore, [113] developed a framework that uses pre-computed paths for failure recovery to reduce the need for online computation. To this end, three different class of heuristics are used. Firstly, variants of the blind geographic routing (BGR) and *k-shortest paths* algorithms are used to obtain at least two edge-disjoint paths. Subsequently, those paths that satisfy requirements are selected for implementation. However, it was found that BGR can generally only provide two paths i.e. only one backup path. Secondly, *shortest-path first* algorithms are combined with a link cost function based on link cost as well as link performance. Thirdly, machine learning algorithms have been combined with search heuristics to obtain reactive search strategies. While this strategy continuously improves results, it is computationally expensive and it cannot be guaranteed that all requirements are satisfied during iteration.

In addition, [125] investigated resilience against single link failure while considering the TCAM limit. This work compared two heuristics: forward local rerouting and backward local routing. The former computes backup paths for each of the links in the primary path. If a failure occurs, the path with the smallest number of additional switches is selected. The latter only computes one backup path, but one that is both node and link disjoint. For benchmarking, the authors also formulated an optimisation problem for minimal cost (based on switch memory and bandwidth consumption of the backup path).

Beyond that, the dynamic use of pre-defined responses to address security incidents has also been studied previously e.g. [75, 63]. However, the focus of these works is on selecting the most suitable response rather than obtaining candidate responses that satisfy given requirements.

In [74], a framework is presented that abstracts network optimisation constraints based on path properties. While near-optimal solutions can thus be obtained much more quickly than using standard optimisation, excluding single causes of failure or considering timing constraints is complex in such a setup.

In his PhD thesis, [4] proposes an algorithm for resilience against single link/node failure based on shortest paths using Dijkstra's or the Bellman-Ford algorithm [18]. However, this work does not consider traffic constraints.

Objective Functions: In addition to the heuristics presented above, various authors also studied optimisation functions. There are several different optimisation goals that can be identified from literature. These will briefly be described in the following paragraphs.

By far the most commonly studied objective function is *minimum cost.* For example, the authors of [139] investigate failure recovery in a resilient SDN network by performing run-time optimisation using iterative routing of feasible solutions until the minimum cost is obtained. However, the only constraint in their formulation is the link capacity, while traffic requirements are not taken into account.

A comparable approach is described in [13]. While not employing an iterative method, the authors focused on minimising operational cost from the available shortest paths in failure cases.

Along similar lines, [76] investigates a delay-constrained routing problem minimising the cost while not considering resilience.

In [137], the minimum cost problem is extended by also taking the facility placement cost into account i.e. where the traffic sources and demands are placed. While dynamical changes of traffic routing and demands are considered, delay-sensitive traffic is not.

In contrast, objective functions that aim to *minimise path length* have been studied less frequently. Among those, [147] focuses on minimising path length of packet forwarding and switch memory usage under the constraint of forwarding table entries for non-resilient networks. Failure recovery that minimises path length has been studied by [27]. This study found the backup path to generally be at least twice as long as the primary path.

Another objective function of interest for critical systems is *minimum delay.* Amongst the works studying this problem, [28] minimises the delay of the flow with the highest delay bound in the network using shortest path and greedy algorithms as well as iterative versions thereof. This is done in the context of using deterministic network calculus during the design phase of safety-critical systems.

Especially for large networks, optimising for a *minimum amount of forwarding rules per switch* has been studied due to the impact on TCAM memory that also affects power consumption. Publications [188] and [91] are examples of this approach. While the former aims to automatically and optimally place rules for a distributed firewall functionality, constrained by the maximum capacity of the switches, the latter investigates fast-failover/multipath routing without rule migrations by optimising rule sharing.

In [8] the *link utilisation is minimised.* The authors use a fully polynomial time approximation scheme rather than linear optimisation due to the faster runtime required for deploying the mechanism online.

Beyond that, [88] introduce a method to find the *maximum resilience* under capacity constraints and scaled rerouting which is then maximised. While the paper takes capacity constraints as well as serving all users into account, latency is not considered. What is more, no hard guarantees are made with respect to resilience. Consequently, the proposed approach is not suitable for critical systems.

Some authors have also compared the effect of *different objective functions* on network performance. Amongst them, [16] used linear programming (LP) to compare various objective functions from literature for the non-resilient case. For evaluation of the respective functions, parameters for link utilisation as well as weighted mean queuing and transmission delay were used. It was found that shortest path algorithms result in a high maximum link utilisation. In the investigated cases, the occurring delay was negligible because of high link capacity. However, their results are not applicable to safety-critical systems, since they allow arbitrarily splitting one flow across different paths through the network, which reduces resilience.

Additionally, [70] compared various heuristic objective functions for their effect on link utilisation and average path length. The investigated functions are either based on cost (including an extension to Fortz' algorithm [57]) or on link utilisation and have been analysed both for single-link failure resilience and for the non-resilient case. This work also found that shortest paths algorithms significantly increase link utilisation, while those heuristics that take link utilisation into account yield longer paths, on average. Comparable effects have been reported for both the resilient and the non-resilient case. However, the effect on the timing behaviour of the traffic demands has not been studied.

2.5 Summary

This chapter has highlighted the necessity of using new technologies to address safety and security issues in critical systems. The key constraints affecting such systems (availability, resilience, timing/predictability) have been presented.

Furthermore, the key ideas behind SDN as well previous efforts to use SDN in critical systems have been described for both data and control plane. The differences between traditional networking and SDN have been illustrated.

In addition, failure recovery from both intentional (i.e. attacks) and unintentional failures has been discussed. Here, the four basic steps of *system monitoring, anomaly detection, impact assessment* and *effect mitigation* to address failures in safety-critical systems have been described. A detailed literature review has been performed.

However, important research topics remain that are not addressed in current literature. Firstly, to the best of the author's knowledge, no comprehensive comparison between different algorithms to obtain alternate paths that mitigate network failures has been performed.

Secondly, published research only focuses on some of the constraints that affect safety-critical systems. However, to provide safety-critical networks, all aspects need to be considered to prevent traffic loss.

Thirdly, the safe rollout of configuration changes in safety-critical systems has not been studied in detail.

Yet without addressing these issues, it is not possible to react to component failures by dynamically adapting the network. However, this capability is essential to address the increasing risk that safety-critical systems face through the exposure to cyber threats.

Consequently, this thesis will study these topics in detail to address these challenges. More specifically, it will be investigated how SDN can enable safety-critical networks to react dynamically and safely to failures.

Chapter 3

System Model

This chapter provides information on how network configurations can be obtained in safety-critical systems. The first section in this chapter, section 3.1 provides the mathematical model used in this thesis. Subsequently, the network model, as well as corresponding constraints and the formulation of optimisation functions is presented in section 3.3. Furthermore, the algorithms for two heuristics used for comparison with optimisation functions are provided.

3.1 General System Model

This section develops the overall system model used in this thesis. In addition, it introduces the concept of appropriateness, where the severity of a reaction is matched to that of an incident.

3.1.1 Basic System Model

As described in the previous chapter, in order to react to failures appropriately, the following components are necessary:

- Sensing
- Anomaly detection
- Impact Assessment
- Mitigation

This corresponds to the standard implementation of a closed-loop control system as depicted in figure 3.1. For a network within the context of safety-critical systems, the plant corresponds to the network, while the control parameters are network properties such as latency, resilience and resource usage. The sensing element corresponds to SDN controllers and potentially additional intrusion detection and/or IVHM. Finally, the possible control input consists of network configurations, as well as security responses such as blacklisting, increased logging frequency etc. This relationship is depicted in figure 3.2.

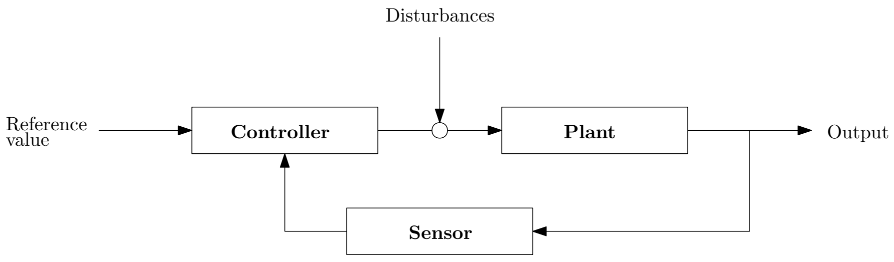


Figure 3.1: Closed-loop control system

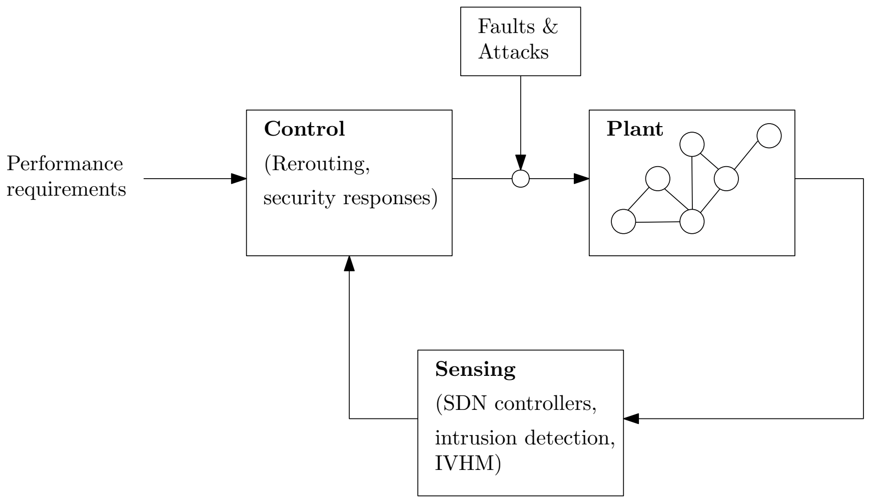

Faults & Attacks
Performance requirements
Control
(Rerouting, security responses)
Plant
Sensing
(SDN controllers, intrusion detection, IVHM)

Figure 3.2: System overview

For safety-critical systems, two properties are of interest to determine the system's state: safety and security. Consequently, the system can be in one of the following four states at any given time:

- s_1: Safe and secure
- s_2: Safe and not secure
- s_3: Not safe and secure
- s_4: Not safe and not secure

State transitions can occur either because of component failures or because of attacks. It should be noted that security incidents can have an impact on safety, e.g. by causing a buffer overflow at a network switch.

Consequently, the system is modelled as a Finite State Machine as a tuple $(\Sigma, S, s_0, \delta, F)$ with the state-transition function δ defined as follows:

$$\delta : S \times \Sigma \to S \tag{3.1}$$

where Σ is the input set and S is the set of states so that $s_1, s_2, s_3, s_4 \in S$ with the initial state $s_0 \in S$. It is assumed that the system is safe and secure upon initialisation, i.e. $s_0 = s_1$. Finally, the set of final states is denoted by F.

The relation between the system states and the impact on the system is shown in figure 3.3. Here, the colours denote the effect on the system. In the context of safety-critical systems, lack of security is considered to be less critical than lack of safety as long as critical functions are not negatively affected.

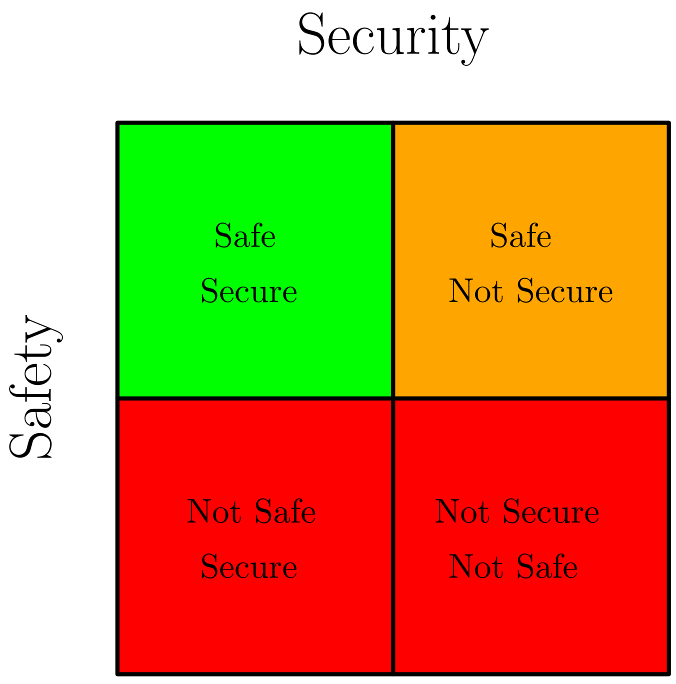


Figure 3.3: Possible system states

3.2 Appropriateness

It is clearly undesirable that a minor change within the network state s_1 results in a complex reconfiguration of the entire network. Consequently, the term of *appropriateness* is defined here.

In order to avoid minor traffic delays in less critical systems affecting safety-critical systems, it is necessary for the severity of the response to be proportional to the severity of a detected anomaly as follows:

$$m \in \mathcal{M} \propto s \in \mathcal{S} \tag{3.2}$$

where $\mathcal{M}$ is the set of possible mitigations and $\mathcal{S}$ the possible network states.

First of all, the impact of the detected anomaly on the network needs to be established. To this end, the network state s is considered to be a tuple that consists of a safety state J and a security state I.

The safety state J is composed as follows:

$$J = \begin{pmatrix} \mu\,\Delta t \\ |d_a| \\ |d_a(s)| \\ c_{ij} \\ c_v \end{pmatrix}, \tag{3.3}$$

where $\mu\,\Delta$ is the average time delay experienced by the flows, d_a are affected non-critical flows, while $d_a(s)$ are affected critical flows. c_{ij} and c_v are the link and switch capacity, respectively.

To complement this, the security state I consists of

$$I = \begin{pmatrix} \text{Integrity} \\ \text{Privacy} \\ \text{Trustworthiness} \end{pmatrix}. \tag{3.4}$$

While common security models also include availability, this factor has already been included in the safety state. Consequently, it is not necessary to consider it again. The safety state J is only applicable to the data plane, as safety-critical traffic can be affected here. In contrast, the security state J exists separately for the data, control and management plane, as security incidents on each plane have different ramifications on the network state. Consequently, as the impact of a security incident is different depending on which plane it occurs on, the corresponding subset of mitigation methods m, differs, demanding separate treatment. In addition, a safety-related incident can also have an impact on the security state. However, as mentioned before, this impact is considered to be less significant that security-related incidents that impact safety.

The same considerations that have been deliberated above for detected anomalies also apply for the reaction. However, unlike the classification of anomalies which are based on the *measured* network states, the classification of reactions can only be based on the *potential* impact. Here it is necessary to also include a feedback loop to make sure that the impact estimation is as accurate as possible.

3.2.1 Effect on SDN Planes

As this thesis considers a SDN network, these states themselves apply to the three different planes: *data, control* and *management plane.* This will be detailed in the following paragraphs.

Firstly, both anomaly and mitigation can have impact on the data plane, e.g. because a path has to be rerouted across a slightly more used link. While configuration templates guarantee that the delay bounds hold, increased delays are possible. In addition, resource usage might also be affected. This gives the impact of a mitigation effort E_m on the data plane $\mathcal{P}$ as a tuple of queuing delay as well as link and switch capacity. Besides, in in-band control networks, the bandwidth of the communication between controller and switches also needs to be taken into account, yielding

$$\mathcal{E}(d) = \begin{pmatrix} \Delta t \\ c_{ij} \\ c_s \\ \mathrm{bw}_{\mathrm{of}} \end{pmatrix}. \tag{3.5}$$

Thirdly, the control plane is also affected by the mitigation effort because forwarding table entries need to be modified. Consequently, the effect on the control plane can be described by the cardinality of the set of flows D_r affected by the reconfiguration and by the cardinality of the set of new forwarding tables R_r that need to be installed:

$$\mathcal{P}(c) = \begin{pmatrix} |d| \\ |r| \end{pmatrix}. \tag{3.6}$$

Finally, the effect on the management plane is more complex to describe. Here, the removal or addition of a component has an impact on the time t_c, complexity l_c, and amount of traffic bw_c that needs to be exchanged to reach a consensus, as well as the impact on resilience R in the management plane. This gives

$$\mathcal{P}(m) = \begin{pmatrix} t_c \\ l_c \\ \mathrm{bw}_c \\ R \end{pmatrix}. \tag{3.7}$$

Notwithstanding, to ensure that the reconfiguration is appropriate to the current network state, the possible reactions are divided into four different subsets, corresponding to the four basic network states (s_1, s_2, s_3, s_4) above. Consequently, while only minor modifications are permissible when the system is safe and secure (i.e. in state s_1), with an increasingly degrading network state more and more reactions become available, until the full reaction set is permissible when the network is neither safe nor secure (s_4).

Formally, this can be described as follows:

$$m = \begin{cases} \begin{pmatrix} n_1 \\ 1 \end{pmatrix} \in \mathcal{M}_1 \subseteq \mathcal{M} & \text{if } S = s_1 \\ \begin{pmatrix} n_2 \\ 1 \end{pmatrix} \in \mathcal{M}_1 + \mathcal{M}_2 \subseteq \mathcal{M} & \text{if } S = s_2 \\ \begin{pmatrix} n_3 \\ 1 \end{pmatrix} \in \mathcal{M}_1 + \mathcal{M}_2 + \mathcal{M}_3 \subseteq \mathcal{M} & \text{if } S = s_3 \\ \begin{pmatrix} n_4 \\ 1 \end{pmatrix} \in \mathcal{M} & \text{if } s = s_4 \end{cases} \tag{3.8}$$

Consequently, any reaction m can be selected from a subset consisting of n reactions forming subsets $(\mathcal{M}_{1..4})$. While designing reactions, the expected impact $E_{m_{\text{expected}}}$ has to be established on the parameters described above. Subsequently, they can be matched into the corresponding reaction subset $\mathcal{M}$.

This ensures that the mitigation efforts are proportional to the impact of the detected anomaly, as required by equation (3.2).

Thus, we obtain associated properties for each such mitigation m. The potential impact P with respect to improvements to safety P_{safe} and security P_{sec} so that

$$P = \begin{pmatrix} P_{\text{safe}} \\ P_{\text{sec}} \end{pmatrix}.$$

3.3 Problem Formulation

After describing the modelling of the overall system states in the previous section, the model of the plant will be detailed. For the purpose of this thesis, we consider the plant to be a safety-critical network. The network model used in the remainder of this thesis in section 3.3.1.

3.3.1 Network Model

In this work, networks are modelled as a graph $G = \{V, E\}$ that consists of nodes V that correspond to the network switches. Each node $v \in V$ has a throughput c_v as well as a number of entries to the forwarding table r_v associated with it. The links in the network $e(i, j) \in E$ are considered to be directed edges from switch i to switch j with $i, j \in V$. Each such link has a corresponding capacity c_{ij} and cost w_{ij} of using this link. Generally, the associated properties are considered to be independent, so that $c_{ij} \neq c_{ji}$ and $w_{ij} \neq w_{ji}$.

This network is used to transport the traffic demands $d \in D$ from a source s connected to switch $a \in V$ to a destination t connected to switch $b \in V$. Each such demand consists of a required bandwidth bw_d as well as the maximum delay $\Delta t_{\max d}$ that can be tolerated until the traffic must have arrived at its destination. In addition, k_d node[1] and link-disjoint paths must be provided, where $k_d \geq 2$. This ensures that no single cause of failure in the network causes the traffic to be lost.

However, as sources/destinations cannot be easily connected to more than one switch, it is assumed that failures of those nodes are mitigated through other means, e.g. several instances of critical computers. Such means are already incorporated in D and hence they do not need to be considered separately in the model. Consequently, each demand is a tuple that consists of the following components: $d = (a_d, b_d, \mathrm{bw}_d, \Delta t_{\mathrm{max}d}, k_d)$.

In addition, three properties of the network need to be taken into account. Firstly, the memory on a SDN switch is limited [91]. Consequently, only a limited number of forwarding rules $r_{\max}$ can be stored at each switch. The number of rules r stored also affects the power consumption and hence heat that needs to be dissipated, which is an important consideration for mobile platforms such as airplanes or other vehicles. Secondly, each switch can only handle a limited number of traffic at any given time.

[1] $v \neq a, b$

This results in each switch having a maximum capacity $c_{\max}$. Thirdly, a similar limit applies for links, giving the maximum bandwidth $bw_{\max}$ for each link.

Thus a software-defined network consisting of a number of network switches, with a given topology and links between them is considered. It is assumed that traffic demands are known beforehand and do not change dynamically during operation. Hence, the following demand set consists of the maximum traffic that needs to be served concurrently. Across this network a number of demands needs to be routed. This network consists of network switches $s \in S$, with the number of forwarding rules r_s and the throughput of the switch c_s. The network links $(i, j) \in L$ have a capacity $c[i, j]$ of the link (i, j) and cost $a[i, j]$ of using this link. This network is used to satisfy the demands contained in D. As this thesis is concerned with safety-critical traffic, all demands in the set need to be met by the network.

Beyond the requirements originating from the traffic demands, three constraints apply. Firstly, the maximum number of forwarding rules r_{max} that may be placed at each switch. Secondly, the maximum capacity c_{max} that each switch can handle. Thirdly, the maximum bandwidth bw_{max} of each link.

Several variables need to be introduced to account for the routing of each such demand. The node sequence $P_{r,d} = \{e(a, m), \ldots, e(n, b)\}$ denotes the r-th disjoint path through the network that the respective traffic demand has to transverse. Here, $r \in R_d = \{1, \ldots, k_d\}$ enumerates the number of resilient paths required.

Beyond that, the binary flow variable $x[r, d, i, j] \in X$ indicates whether a specific link is used by the demand on its r-th path:

$$x[r, d, i, j] = \begin{cases} 1 & (i, j) \in P_{r,d} \\ 0 & (i, j) \notin P_{r,d} \end{cases} \tag{3.9}$$

Multipath routing would reduce the available resilience, as fewer disjoint paths are thus available. Consequently, it is not considered in this model.

For safety-critical and real-time traffic, another important factor is the delay experienced by a flow. The delay model proposed in this thesis follows the definition described in [73] and uses

$$\Delta\mathrm{t} = \sum_{e_{i,j} \in E} (t_{\mathrm{pp}} + t_{\mathrm{t}}) \sum_{v \in V} (t_{\mathrm{pr}} + t_q[d]) \tag{3.10}$$

i.e. the sum of propagation t_{pp}, transmission t_t, processing t_{pr} and queueing t_q. Generally, the most significant factor is the queueing delay, while propagation, transmission and processing delay depend on fixed, physical properties of the respective hardware.

To linearise the multiplication of the variable x to consider queueing delay, an auxiliary variable y is introduced so that it will take the following values depending on whether the flow is placed across a particular link:

$$y[r,d,i,j] = \begin{cases} \sum_{d \in D} \sum_{r \in R_d} \frac{x[r,d,i,j] \cdot \mathrm{bw}_d}{c_{\max}(i)} & (i,j) \in P_{r,d} \\ 0 & otherwise \end{cases} \tag{3.11}$$

Here, the worst-case of all paths being served concurrently is used. This ensures that there is no violation of the real-time traffic in any possible combination of applied configurations. Hence the queueing delay experienced by a flow can be determined

$$\Delta t_q[r,d] = \sum_{e(i,j) \in P_{r,d}} y[r,d,i,j] \tag{3.12}$$

by summarising the values of y over all links that the flow is transversing.

3.3.2 Constraints

As this network needs to transport safety-critical traffic, there are several constraints that apply to the problems detailed above. Firstly, to ensure resilience all demands have to be routed k times

$$\forall d \in D: \sum_{r \in R_d} \sum_{e(d_a,j) \in E} x[r,d,a_d,j] = k_d \tag{3.13}$$

from the node a to which the source is attached. The same applies to all requests that need to arrive at the node b that is connected to the destination:

$$\forall d \in D: \sum_{r \in R_d} \sum_{e(i,d_b) \in E} x[r,d,i,b_d] = k_d. \tag{3.14}$$

To guarantee that all flows are forwarded between those endpoints, incoming flows and those originating at the switch need to be forwarded:

$$\forall r \in R_d, \forall d \in D, \forall v \in V: \sum_{e(j,v \neq b_d) \in E} x[r,d,j,v] = \sum_{e(v \neq a_d,j) \in E} x[r,d,v,j]. \tag{3.15}$$

This does not apply to flows that are destined for the respective switch. Furthermore, the maximum available capacity $c_{\max}$ at the node

$$\forall v \in V: \sum_{r \in R_d} \sum_{d \in D} \sum_{e(v,j) \in E} x[r,d,v,j] \cdot d_{\mathrm{bw}} \leq c_{\max} \tag{3.16}$$

and the maximum capacity bw_{max} at each link

$$\forall e(i,j) \in E : \sum_{r \in R_d} \sum_{d \in D} x[r,d,i,j] \cdot d_{\text{bw}} \leq \text{bw}_{\text{max}}[i,j] \tag{3.17}$$

must be less than the combined bandwidth demands of all flows using the given component. Formally, as this model considers directed links, the capacity limit applies to $e(i,j)$ and $e(j,i)$ independently.

Additionally, as has been detailed above, the TCAM in the switches is limited. Hence, the number of outgoing flows needs to be less than the maximum number of forwarding table entries that can be contained in the memory:

$$\forall v \in V : \sum_{r \in R_d} \sum_{d \in D} \sum_{e(v,j) \in E} x[r,d,v,j] \leq r_{\text{max}}. \tag{3.18}$$

Beyond that, the delay experienced needs to be less than the delay $\Delta t_{\text{max}\,d}$ permissible for this flow:

$$\sum_{e(i,j) \in E} y[r,d,i,j] \leq \Delta t_{\text{max}\,d}. \tag{3.19}$$

Furthermore, as resilience is a key requirement for safety-critical systems, equation (3.20) guarantees that at most one flow uses a specific link, thus ensuring that no links are shared between resilient flows:

$$\forall d \in D, \forall e(i,j) \in E : \sum_{r \in R_d} x[r,d,i,j] + \sum_{r \in R_d} x[r,d,j,i] \leq 1, \tag{3.20}$$

Likewise, equation (3.21) guarantees that no switches are shared between resilient flows, safe the origin and destination. As mentioned above, failures of a_d and b_d need to be addressed through the demand set.

$$\forall d \in D, v \in V : \sum_{r \in R_d} \sum_{e(v,j) \in E} x[r,d,v,j] \leq \begin{cases} k_d & \text{if } v \in \{a_d, b_d\} \\ 1 & \text{otherwise} \end{cases} \tag{3.21}$$

All resilience constraints are derived from the application of SDN to safety-critical traffic. In equation (3.20), the pessimistic case of a physical failure affecting both flow directions is assumed. Yet where applicable, different formulations may easily be made.

3.3.3 Optimisation Problem

As described in chapter 2, there has been no detailed study to date that compares various algorithms to obtain routings to mitigate the effects of network component failures. To ensure that safety-critical demands are satisfied, it is necessary that the constraints as addressed above (equation (3.13) until equation (3.21)) are considered. Consequently, this gives the following optimisation problem:

$$\begin{aligned}
&\text{minimize} \quad f \\
\textit{s.t.} \qquad & \forall d \in D: \sum_{r \in R_d} \sum_{e(d_a,j) \in E} x[r,d,a_d,j] = k_d \\
& \forall d \in D: \sum_{r \in R_d} \sum_{e(i,d_b) \in E} x[r,d,i,b_d] = k_d \\
& \forall r \in R_d, \forall d \in D, \forall v \in V: \sum_{e(j,v \neq b_d) \in E} x[r,d,j,v] = \sum_{e(v \neq a_d,j) \in E} x[r,d,v,j] \\
& \forall v \in V: \sum_{r \in R_d} \sum_{d \in D} \sum_{e(v,j) \in E} x[r,d,v,j] \cdot d_{\mathrm{bw}} \leq c_{\max} \\
& \forall e(i,j) \in E: \sum_{r \in R_d} \sum_{d \in D} x[r,d,i,j] \cdot d_{\mathrm{bw}} \leq \mathrm{bw}_{\max}[i,j] \\
& \forall v \in V: \sum_{r \in R_d} \sum_{d \in D} \sum_{e(v,j) \in E} x[r,d,v,j] \leq r_{\max} \\
& \sum_{e(i,j) \in E} y[r,d,i,j] \leq \Delta t_{\max d} \\
& \forall d \in D, \forall e(i,j) \in E: \sum_{r \in R_d} x[r,d,i,j] + \sum_{r \in R_d} x[r,d,j,i] \leq 1 \\
& \forall d \in D, v \in V: \sum_{r \in R_d} \sum_{e(v,j) \in E} x[r,d,v,j] \leq \begin{cases} k_d & \text{if } v \in \{a_d, b_d\} \\ 1 & \text{otherwise} \end{cases}
\end{aligned} \tag{3.22}$$

In the problem statement above, f can be one of the following five different primary objective functions: *minimum cost* as provided in equation (3.23), *minimum queueing delay* as given equation (3.24), *minimum number of forwarding table entries* described in equation (3.25), *maximum resilience* as provided in equation (3.26) as well as a variant on *minimum cost* listed in equation (3.27) from literature. In this case, cost is calculated as a function of link utilisation. The resulting routings are investigated and compared in the following chapter.

The optimisation functions f can be expressed as follows with the *minimum cost* as the sum of the costs of using a specific link with the bandwidth required:

$$\min_x \sum_{r \in R} \sum_{d \in D} \sum_{e(i,j) \in E} w_{e(i,j)} \cdot x[r,d,i,j] \cdot d_{\mathrm{bw}}, \tag{3.23}$$

and the *minimum queueing delay* as

$$\min_{x} \sum_{d \in D} \Delta t. \tag{3.24}$$

Beyond that, the *minimum number of forwarding table entries* can be established from the number of flows leaving the respective switches:

$$\forall v \in V \ \min_{x} \sum_{d \in D} \sum_{r \in R} \sum_{e(v,j) \in E} x[r, d, v, j], \tag{3.25}$$

and the *maximum resilience* as

$$\max \sum_{d \in D} d_k. \tag{3.26}$$

Fortz et al. [57] proposed an objective function for *minimum link usage* that uses the link utilisation to derive cost so that links become costlier with increased link usage as follows:

$$w_{e(i,j)} = \begin{cases} 1 & \text{for} \quad 0 \leq x < 1/3 \\ 3 & \text{for} \quad 1/3 \leq x < 2/3 \\ 10 & \text{for} \quad 2/3 \leq x < 9/10 \\ 70 & \text{for} \quad 9/10 \leq x < 1 \\ 500 & \text{for} \quad 1 \leq x < 11/10 \\ 5000 & \text{for} \quad 11/10 \leq x < \infty \end{cases} \tag{3.27}$$

Otherwise, the objective function is identical to equation (3.23).

There may be more than one solution that provides the same value for the primary objective functions given above. To ensure that the comparison with respect to the various performance characteristics is fair, secondary optimisations need to be performed. While the objective functions detailed above are being used again here, three additional secondary objective functions need to be introduced to cover all performance characteristics.

The bandwidth of the links used is minimised as follows:

$$\min_{e} \sum_{r \in R} \sum_{d \in D} x[r, d, i, j] \cdot d_{\text{bw}}. \tag{3.28}$$

A similar formulation is used for the bandwidth used at the switches:

$$\min_{v} \sum_{r \in R} \sum_{d \in D} x[r, d, v, j] \cdot d_{\text{bw}}. \tag{3.29}$$

Additionally, the path length can be minimised thus:

$$\min_{d} \sum_{r\in R} \sum e_{ij} \in Ex[r,d,i,j]. \tag{3.30}$$

The formulation for *minimum cost*, *minimum delay* and *minimum number of forwarding table entries* follows equation (3.23), equation (3.24) and equation (3.25) respectively. In addition to the constraints given above, the result of the primary objective function is given as a constraint so that for both *minimum cost* functions, the constraint is given as

$$\sum_{r\in R}\sum_{d\in D}\sum_{e(i,j)\in E} w_{e(i,j)} \cdot x[r,d,i,j] \cdot d_{\text{bw}} = \min_{x} \sum_{r\in R}\sum_{d\in D}\sum_{e(i,j)\in E} w_{e(i,j)} \cdot x[r,d,i,j] \cdot d_{\text{bw}}, \tag{3.31}$$

for the *minimum delay* as

$$\sum_{d\in D} \Delta t = \min_{x} \sum_{d\in D} \Delta t, \tag{3.32}$$

for the *minimum number of forwarding table entries* as

$$\forall v \in V \sum_{d\in D}\sum_{r\in R}\sum_{e(v,j)\in E} x[r,d,v,j] = \forall v \in V \ \min_{x} \sum_{d\in D}\sum_{r\in R}\sum_{e(v,j)\in E} x[r,d,v,j], \tag{3.33}$$

and for the *maximum resilience* as

$$\sum_{d\in D} d_k = \max_{x} \sum_{d\in D} d_k. \tag{3.34}$$

3.3.4 Heuristics

For comparison with the optimisation functions of the previous section, four heuristics were investigated: the shortest path and Dijkstra algorithm commonly used for routing. While a variety of heuristics for routing have been investigated in literature, most are based on Dijkstra's algorithm (cf. section 2.4) and only include minimal adaptations to suit a specific application.

Hence, two simple heuristics that focus on some aspects of critical systems were additionally investigated. The first considers the network capacity and is abbreviated as *Heur. Capa.* in figures. It is based on Dijkstra's algorithm until the link capacity is exceeded, at which point that link can be no longer used. If no route can be found using links with capacity available, complete topology becomes available again.

In contrast, the second heuristic (*Heur. EDF* in figures) implements a simple earliest deadline first (EDF) schedule, where the path with the smallest resulting latency is selected for placement first and the placement of successive flows must not violate the requirements of the previous ones. For this, all loop-free paths are calculated and the one with the smallest latency selected. The complete algorithm for the EDF heuristic is provided by algorithm 3.1.

These algorithms are used to calculate two paths, where the second one is link-disjoint from the first.

Listing 3.1: Earliest-Deadline First (EDF)

```
# sort demands by timing requirement
sorted_demands=sort_demands_by_timing()
# iterate over all demands
for d in sorted_demands:
 # obtain all loop-free paths between
 # source and destination node:
 get_all_simple_paths(source, destination)
 # calculate the resulting latency of all
 # paths based on the link usage:
 calculate_latency(link_usage)
 # order the paths by ascending
 # latency:
 ordered_paths=sort_by_latency()
 # make lowest latency path
 # default path and next
 # backup path:
 default=ordered_paths[0]
 backup=ordered_paths[1]
 # add selected paths to link
 # usage:
 update_link_usage(default, backup)
```

Listing 3.2: Capacity-constrained heuristic

```
# iterate over all demands
for d in demands:
 # obtain shortest path between
 # source and destination node,
 # with standard topology
 default=get_shortest_path(topo, source, destination)
 # calculate backup path
 # remove p from topology
 topo2=topo-default
 backup=get_shortest_path(topo2, source, destination)
 # add selected paths to link
# usage:
update_link_usage(default, backup)
# remove link from topology if capacity
# is exceeded:
if link_usage[link]>=max_capacity:
  topo-=link
  topo2-=link
```

3.3.5 Summary

This chapter presented the network model used. Subsequently, both general networking constraints (demand placement, flow continuity, capacity limits of links and switches) and constraints specific to the use of SDN in critical systems (TCAM capacity, resilience, delay bound) were discussed.

In addition, the formulation of five primary objective functions (*minimum cost*, *minimum queueing delay*, *minimum number of forwarding table entries*, *maximum resilience* as well as a variant on *minimum link utilisation*) has been given.

For comparison with these objective functions, two heuristic algorithms (EDF and constrained capacity) are introduced and discussed. Both are based on Dijkstra's shortest path algorithm, but either route the demands with the smallest delay bound first or remove links from the network if they are exceeding capacity.

Chapter 4

Generating Configuration Templates

In order to find the most suitable way to generate network configurations, the effect of different algorithms and objective functions on network behaviour has to be investigated. While heuristics can provide results quickly and require fewer computational resources than optimisation, results from heuristics are harder to predict. Consequently, this chapter investigates whether quality criteria (cf. section 4.1) can be satisfied and whether hard constraints are violated. This allows to find which algorithms are suitable to calculate resilient paths in safety-critical systems.

To this end, the analysed network parameters are detailed and an overview about the network topologies and traffic characteristics of the studied networks is provided.

Subsequently, the results of using the heuristic methods and optimisation functions presented in section 3.3 to obtain network configurations are provided. Next, a comparison between the results obtained through heuristics and optimisation is performed and the main results summarised. This chapter concludes with describing how the network configurations calculated can be used to react to detected network failures in section 4.6.

4.1 Parameters Investigated

In this chapter, the effect of the following parameters (ordered alphabetically) is compared:

- Bandwidth of
  - Links
  - Switches

- Calculation time
- Cost
- Delay
- Forwarding table entries
- Memory usage during calculation
- Path length

The bandwidth used by deploying various routing algorithms has been investigated, because it provides information on how well they could scale to future demands, which may require more bandwidth.

To a limited degree, the same is also true for the calculation time, however the $\mathcal{O}$ values are generally known and provide sufficient information on complexity. However, it is of interest to provide an insight into whether such calculations can be performed online or need to be performed offline prior to deployment.

Small variations in cost are generally negligible for critical systems, where the focus is being placed on safety. Notwithstanding, significant differences do matter, hence it is studied here.

Of greater importance is the delay that the traffic demands experience. As real-time traffic is constrained by delay bounds, whether or not they can be achieved is an important selection criterion for a candidate algorithm.

The usage of the TCAM memory by the number of forwarding table entries has been investigated due to its effect on power consumption [188]. On mobile safety-critical networks such as on-board aircraft or cars, heat dissipation caused by this is a significant issue. For the analyses below, it has been assumed that up to 8,000 forwarding rules can be contained in each SDN switch, as current switches can support between 2,000 and 20,000 rules [77, 90, 89, 166].

The rational behind looking into memory usage is to determine whether available resources are sufficient for onboard calculation. As computing resources – like all elements of critical systems – have to undergo a stringent and lengthy certification procedure, they are generally not as powerful as contemporary consumer goods. As mentioned above, heat dissipation is an additional constraint to putting very powerful computing resources in mobile systems or not easily accessible systems.

In contrast, path length is investigated because longer paths make it more likely that a flow is affected by *any* failure, under the assumption that components have similar failure probabilities. Furthermore, it also provides an insight into the distribution of demands across the network and whether considering delay bounds affects this.

This chapter is structured as follows: section 4.2 describes the key features of the network topologies under study, while the comparison between heuristics is given in section 4.3. Subsequently, the different objective functions for optimisation are compared in section 4.4. This chapter concludes with a comparison between the most efficient objective function and the most suitable section 4.5.

Most of the following figures depicting results are presented as boxplots of the respective algorithms. These plots depict the mean as a solid line while quartiles and whiskers indicate statistical distributions. Outliers are shown as dots. When there was no difference between different runs (e.g. for memory usage), dotplots are provided.

If the results have been comparable across different network topologies, the results of only one network will be presented hereafter. The results of all calculations are provided in appendix C.

4.2 Networks Under Study

To compare the effects of algorithm performance, several different network topologies with varying numbers of demands, links and nodes were investigated. Of those, only the avionics network is used for safety-critical traffic. The remaining topologies are frequently used in literature.

4.2.1 Avionics Network

The avionics network used hereafter is based on the description in [33]. It has been designed to represent the onboard network connecting the flight-critical systems of a modern transport category airplane. The main properties of this network are summarised in table 4.1, table 4.2, and table 4.3. From these values, the bandwidth requirement per flow is defined as

$$\forall d \in D : d_{\text{bw}} = \frac{\text{Framelength}}{\text{bandwidth allocation gap(BAG)}} \tag{4.1}$$

The number of virtual local area networks (VLANs) does not match between the tables due to multicast, i.e. the same demand being replicated and routed differently. Hence, the corresponding distribution is used across the number of demands investigated. For the following experiments, the number of demands has been increased in regular intervals between 98 and 1868, i.e. the full demand set as provided in [33]. This allows to study the effect of varying demand sets. As the maximum bandwidth could not be obtained from literature, it has been assumed that both links and switches can use up to 10^9 Bit i.e. one Gigabit.

Table 4.1: Number of Virtual LANs from Source *a* to Destination *b* [33]

a/b	1	2	3	4	5	6	7	8
1		71	78					34
2	72			77				34
3	90			212	35		42	52
4		97	134			37	35	48
5			80			72	64	
6				82	61		52	
7			52	47	59	67		
8	51	45	43	52				

The latency requirements in table 4.4 are based on the information for flight data recording in [49, AMC CAT.IDE.A.190], which gives a good overview of the expected time criticality.

4.2.2 Standard Networks

As mentioned above, four network topology instances that are commonly used in literature (ATLANTA, Di-Yuan, DFN and NOBEL-GERMANY) were investigated with the relevant information (traffic demands, link capacity, link cost) obtained from the `SND-LIB` [134] library in addition to the avionics network described in the previous section. This database contains several network instances, each consisting of the respective topology, network demands as well as link capacities and cost associated with using those links.

The main properties of these instances are provided in table 4.5, while the network topologies are depicted in figure 4.1.

Table 4.2: Bandwidth Allocation Gap

BAG ms	**Number of VLANs**
2	20
4	40
8	78
16	142
32	229
64	220
1280	255

Table 4.3: AFDX Frame Lengths [33]

Frame length (bytes)	**Number of VLANs**
0-150	561
151-300	202
301-600	114
601-900	57
901-1200	12
1201-1500	35
>1500	3

Table 4.4: Probability Distribution for $\Delta t = \Delta t_{\max}[d]$ for Critical Networks

$\mathbf{\Delta t}$	125	25	500	1000	2000	4000	8000
$\mathbf{P(\Delta t)}$ in percent	2.5	7.5	15	50	15	7.5	2.5

However, these topology instances do not consider resilient routing. This imposes additional demands on the network as more traffic has to be transported. Consequently, the highest link capacities available (with the corresponding costs) were selected. As these traffic instances do not include a resilience requirement, $d_k = 2$ has been assumed for the minimisations and heuristics and $d_k \geq 2$ for the *maximum resilience* objective function.

Likewise, this data set does not include values for the maximum permissible delay. To the best of the author's knowledge, no data with respect to common latency distribution has been published, most likely as it is either proprietary for current or unknown for future applications. Hence, it has been assumed that it follows a Gaussian distribution, as very little traffic such as warnings of imminent danger is highly critical. Low priority traffic such as temperature readings are equally rare.

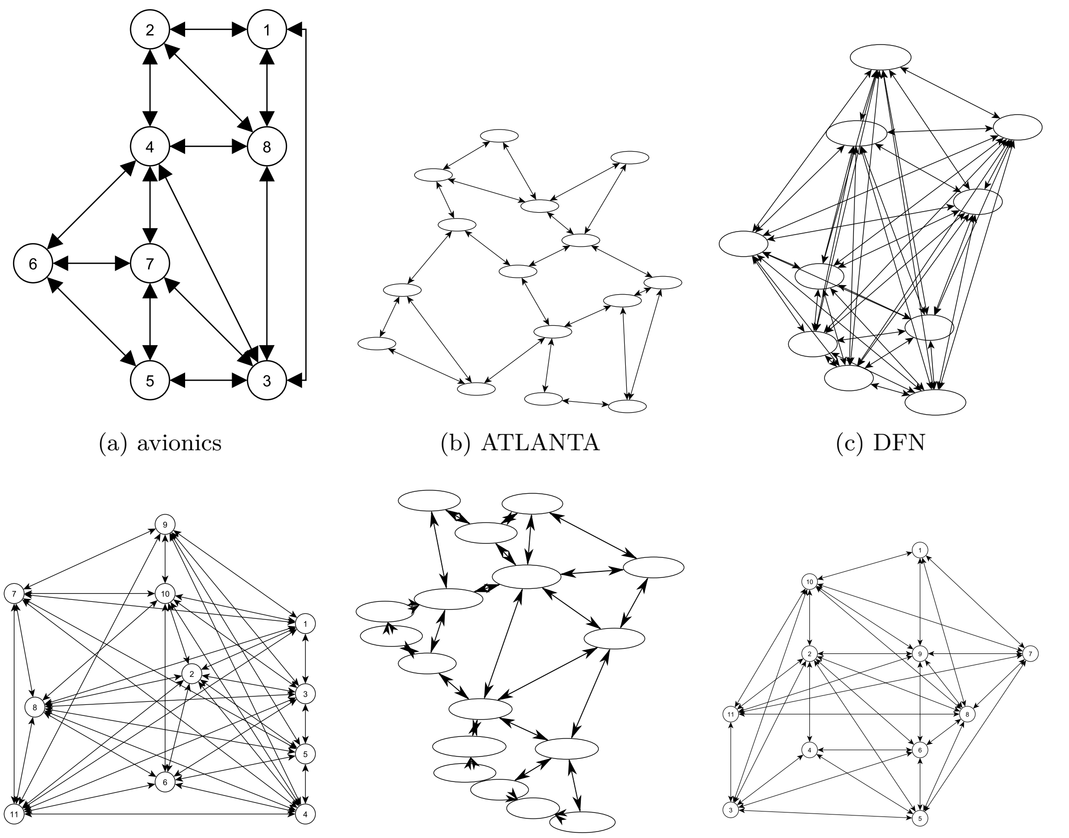
For the following studies, this has been set to have the parameters of $\mu = n \cdot |D|, n = \{1 \ldots 10\}$ and $\sigma = |V|$ for the different topologies, with ten iterations each. If at least one problem for a given μ has been infeasible, the remaining problems were not considered. This also allows to study the effect of varying proportions of extremely time-critical traffic. If results do not differ significantly between the networks under study, only results of one network are presented henceforth. The remaining results can be obtained from appendix C.

Table 4.5: Network Parameters

Network	**Nodes**	**Links**	**Demands**
Avionics	8	15	98-1869
ATLANTA	15	22	210
DFN	10	45	90
Di-Yuan	11	42	22
NOBEL-EU	28	41	200
PDH	11	34	244

4.3 Heuristics

As a benchmark, the shortest path and Dijkstra algorithm of Python's `networkx` software package were used. Additionally, the two heuristics, *Heur. Capa.* and *Heur. EDF* as defined in section 3.3.4 were implemented. Those were selected, as the capacity and



(a) avionics (b) ATLANTA (c) DFN

(d) Di-Yuan (e) Nobel-Germany (f) PDH

Figure 4.1: Network topologies

latency requirements were observed to be frequently violated by applying the Dijkstra algorithm. Since heuristics – unlike optimisation – do not guarantee that all constraints are satisfied, the adherence to constraints is studied for these algorithms.

In this section, only the results for the avionics network and key results from other networks will be presented in the main part of this thesis. The complete results for heuristics for all networks under study can be found in appendix C.1.

Since the latency requirements have been randomly generated (cf. section 4.2), ten problem instances were created and solved for each network.

The key results of this section have been published in [142] and [144]. Both publications only discusses a subset of the performance characteristics given in this thesis. Crucially, [142] does not include results on constraint violations by heuristics. While this is included in [144], this paper only includes comparisons for link usage and TCAM memory usage. Moreover, the papers only gives a very brief summary of the findings, while this thesis provides a systematic and detailed discussion and also provides the results for a more different network. Finally, both papers also include a comparison with an objective function and do not exclusively focus on the comparison of heuristics. Consequently, all of the text contains significant differences and changes compared to the papers.

4.3.1 Bandwidth

Studying the used bandwidth is interesting for two reasons: firstly, it allows to analyse resource usage. Secondly, it permits to judge how many resources remain available so that the network can adapt to future needs with different demand sets. As mentioned above, the bandwidth used has been investigated separately for switches and links.

The resulting switch usage is shown in figure 4.2, while the link usage for the avionics network is depicted in figure 4.3. With respect to the link usage, there is very little difference between Dijkstra and constrained capacity heuristics. This is because the maximum capacity is not reached, consequently the link is not removed from the network and thus, the Dijkstra element of the algorithm is dominant.

In contrast, the minimum link utilisation for the EDF algorithm is about 27.3% higher while the maximum link utilisation is 4.5% lower for 98 demands. This ratio rises to 39.9% and 10.1% for 1869 demands, respectively. Consequently, the higher number of demands with according delay bounds limits the difference between minimum and maximum link utilisation.

With respect to the switch usage, there is again no difference between Dijkstra and constrained capacity heuristics. Yet in this case, the EDF heuristic yields higher switch utilisation (between 71.7% to 36.1% for the minimum and 3.5% to 7.7% for the maximum with increasing number of demands). This is most likely as the algorithm only considers the queueing delay at the link, consequently a less congested link is selected. However, this does not reduce the switch utilisation as the traffic at the switch remains the same. There, avoidance of congested links means that less critical traffic has to accept longer paths, with in turn results in a higher amount of traffic that has to be handled at the switches.

In this network, the maximum bandwidth is not reached by any of the algorithms, consequently no capacity constraints are violated. In addition, while the EDF heuristic displays a slightly higher maximum bandwidth used at the switches, the maximum link usage is slightly higher for the other two heuristics. Hence, the difference between heuristics with respect to utilised bandwidth is not significant to prefer one over the other for the generation of configuration templates.

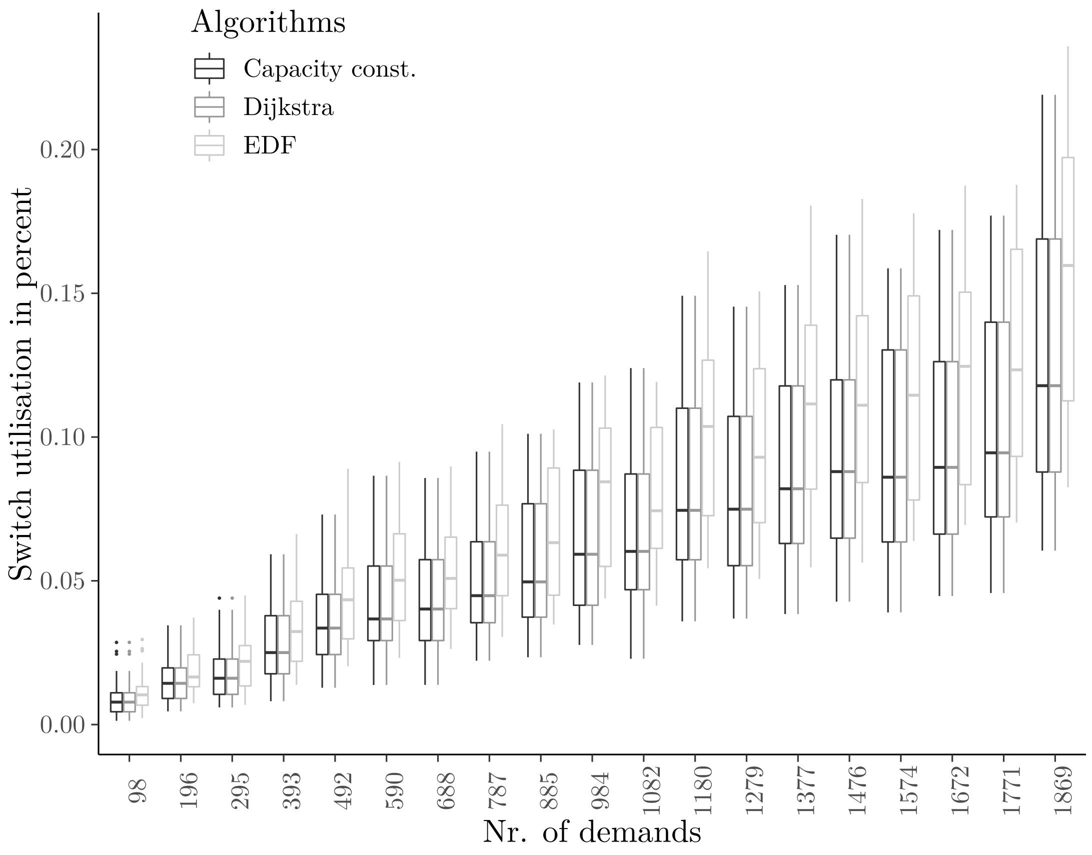


Figure 4.2: Switch utilisation using heuristics for the avionics network

4.3.2 Cost

Figure 4.4 shows that the cost varies between the methods. The shortest paths are more expensive to use for the network topologies investigated, and especially penalised in the `SND-LIB`. Consequently, as the EDF heuristic limits the number of flows on those links by placing only the most critical demands there, this algorithm generates lower cost. It should be noted that for the avionics network this effect could be observed even though the cost of using any one link are identical. Moreover, as the ATLANTA network includes a link that comes close to its capacity limits, the capacity constrained algorithm gives a cost that is about 1.5% lower than that using the EDF algorithm. However, it is interesting to observe that Dijkstra and lim. capacity do not show variance between the different runs for ATLANTA and Nobel-Germany networks.

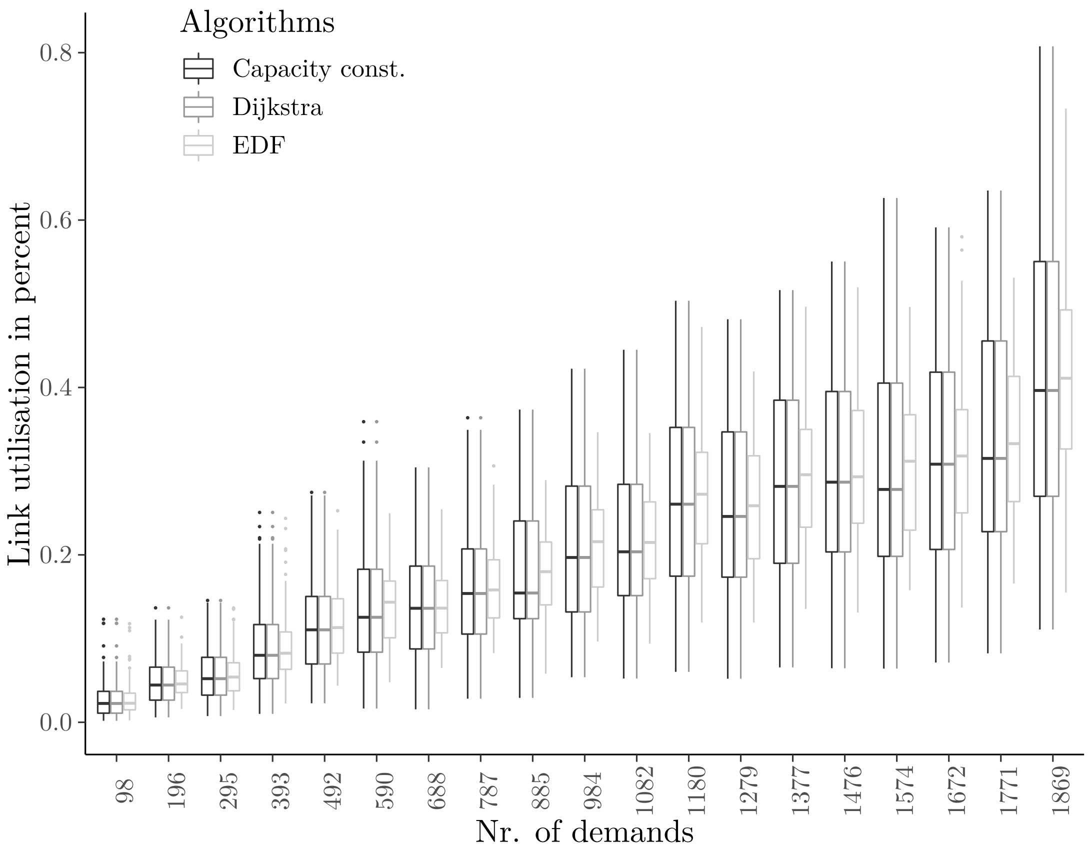


Figure 4.3: Link utilisation using heuristics for the avionics network

While small variations in cost are generally negligible for critical systems, significant differences do matter. However, the differences in resulting cost were small, consequently there is no specific preference of any heuristic algorithm with respect to this aspect.

4.3.3 Time Delay

The time delay Δt of the PDH network only differs in the resilient case. Here, the EDF yields a lower delay for some of the most critical paths, resulting in a lower average delay.

In contrast, for the avionics network, Dijkstra's algorithm results in the smallest mean delay, but also shows the largest variance of the heuristics. From figure 4.5 it can be observed that the maximum queueing delay obtained for the EDF heuristic increases in a stepwise manner, while the others increase linearly. Furthermore, up to 1476 demands, the EDF yields significantly lower queueing delays than Dijkstra's. Above this value, the increased number of demands makes it more likely that a critical demand is placed

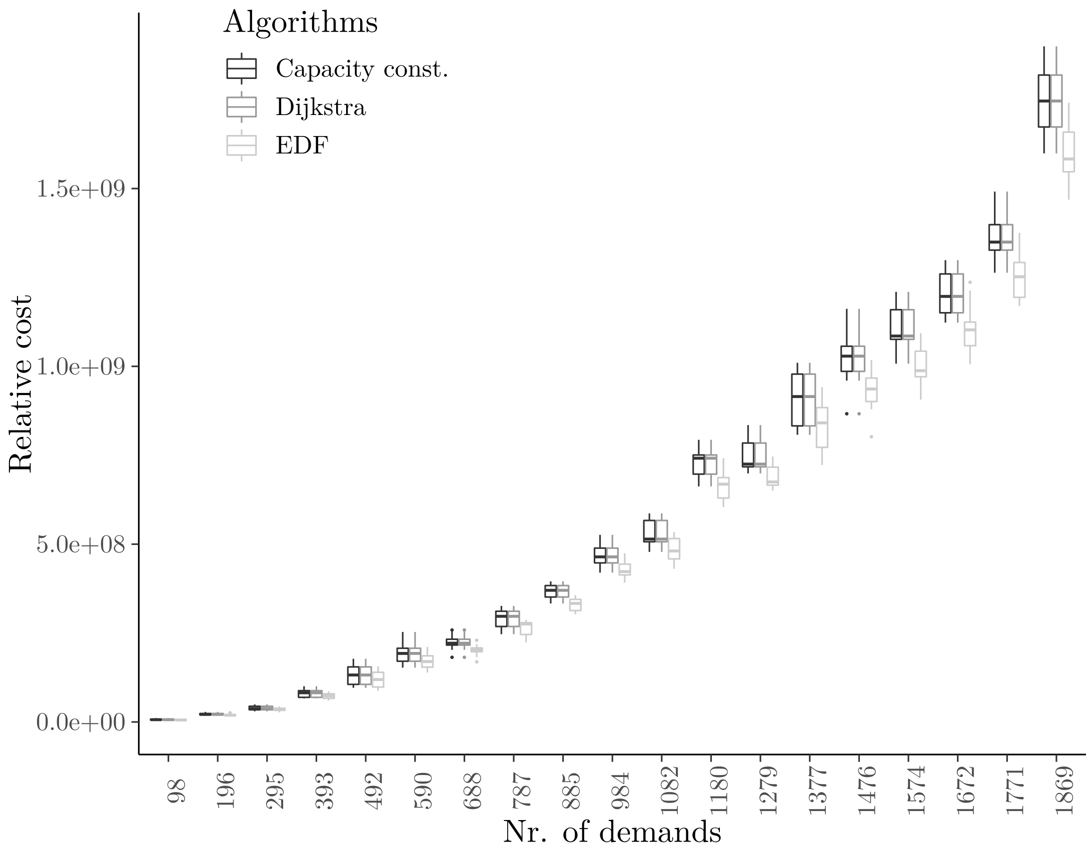


Figure 4.4: Relative cost using heuristics for the avionics network

upon any one link. If violations cannot be prevented, the algorithm returns again to the full topology. Consequently, above this value, the delay bound is no longer a limiting factor, giving higher queueing delay to most demands, and not just the less critical ones.

Here, a low queueing delay is desirable, as it reduces the likelihood of delay bound violations. Moreover, the lower the queueing delay, the more traffic can be added without resulting in violations. Consequently, the EDF heuristic is the most suitable of the investigated heuristics to generate templates. However, all heuristics resulted in violations of the delay bound for the avionics network, which will be analysed in detail below.

To show this effect more clearly, figure 4.7 depicts the density distribution of the resulting latencies. As this network does operate far from capacity limits, the results for the capacity const. heuristic are identical to those of Dijkstra's algorithm, and hence hidden in the plot by the latter. While Dijkstra's algorithm generally results in a lower

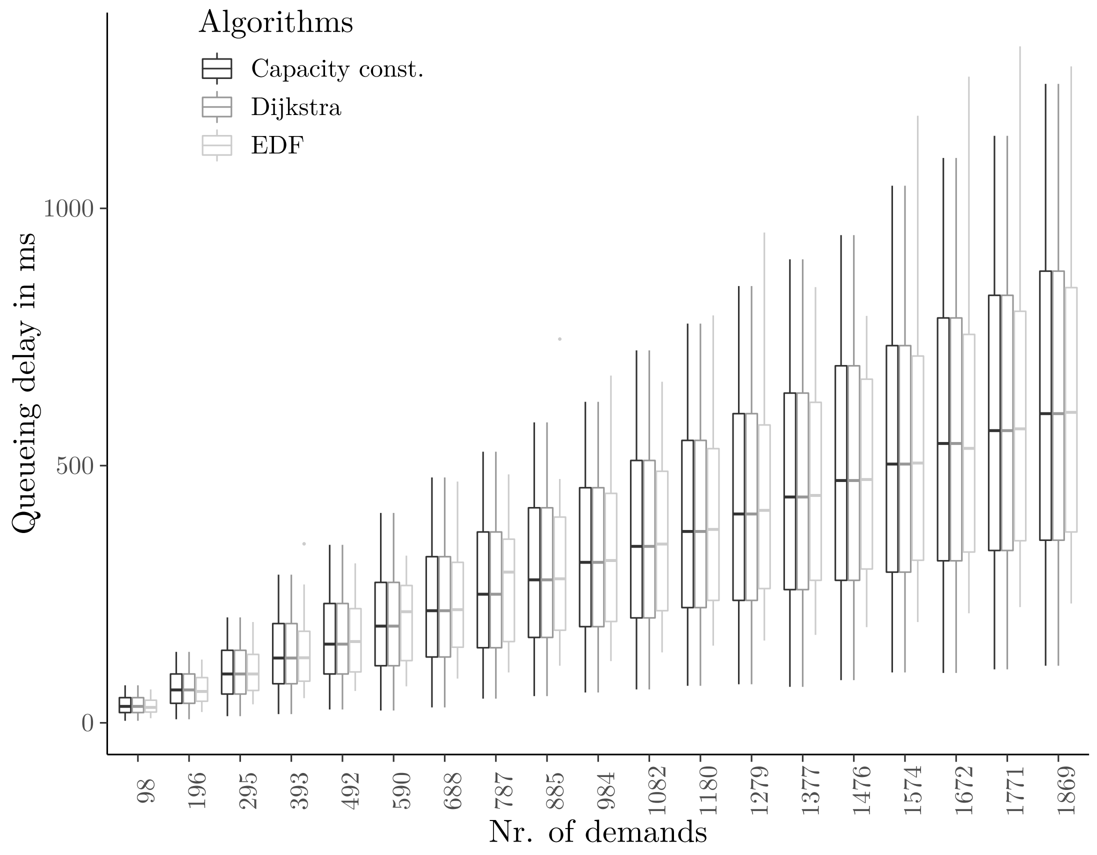


Figure 4.5: Time delay using heuristics for the avionics network

mean latency as well as a smaller variance, the most critical timing constraints are being violated, as shown later. The EDF heuristic yields a smaller variance in the latency and the density has two main peaks rather than Dijkstra's four. Here, the increased delay for the backup path (as the shortest path is already occupied by the primary path) is the cause. As the path with the smallest queueing delay is selected for each demand, the paths are more evenly routed across the network. Yet constraint violations could only be prevented by not placing further demands if they would violate existing, more critical paths.

4.3.4 Forwarding Table Entries

The number of forwarding rules required to be saved on the switches is depicted in figure 4.8. Among the heuristics, Dijkstra's algorithm yields shortest paths, thus fewer rules need to be saved in the forwarding tables of the switches. Taking into account the delay bound, as done by the EDF heuristic leads to longer paths and thus more

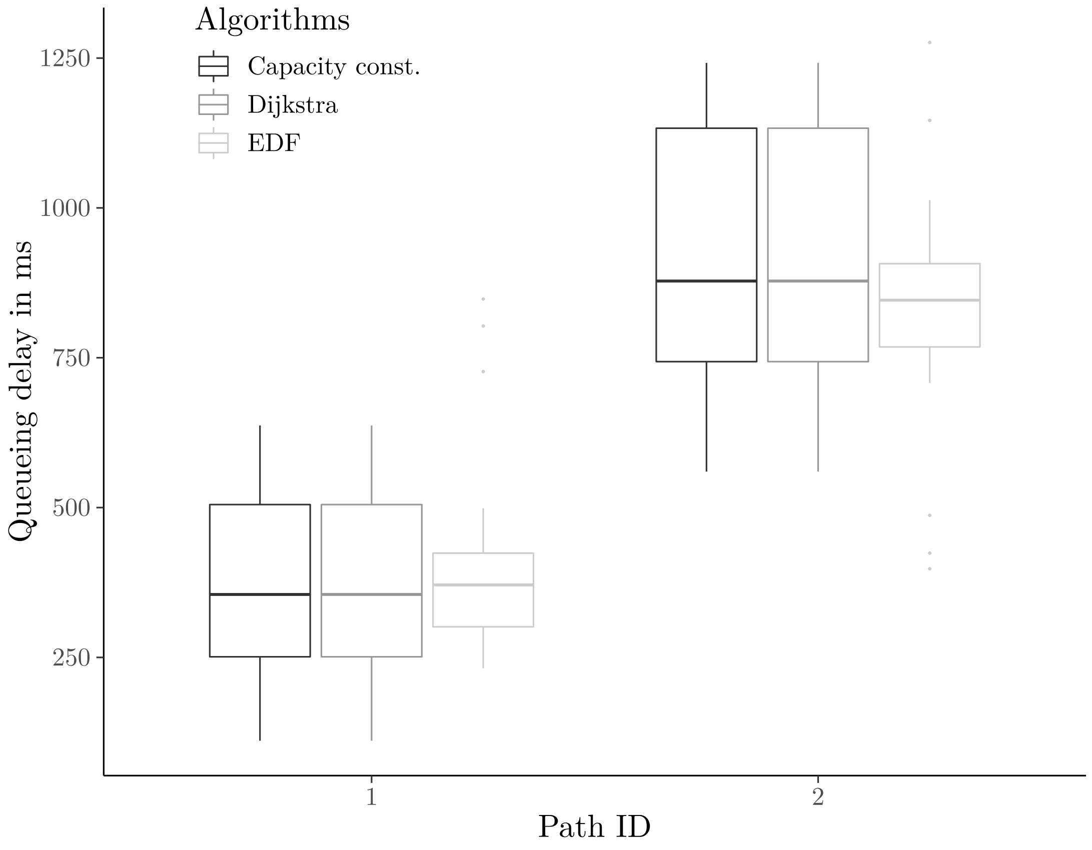


Figure 4.6: Time delay using heuristics for the avionics network for $|D|$=1869

switch memory being used. While literature [91] has proposed to share rules between different demands, as different levels of criticality need to be considered separately to avoid less critical demands affecting more critical ones, the potential benefit its limited for critical systems. Notwithstanding, the EDF algorithm used on the avionics network distributes traffic much more evenly across the network than Dijkstra's. Consequently, the variance between the required memory of the least and most used switch is lower.

Here, the number of forwarding rules to be stored is not only a hard limit imposed by memory constraints but also affects energy consumption and heat dissipation. Consequently, Dijkstra's algorithm is the most suitable for mobile systems, where this is especially relevant. However, the differences between the algorithms is not especially large.

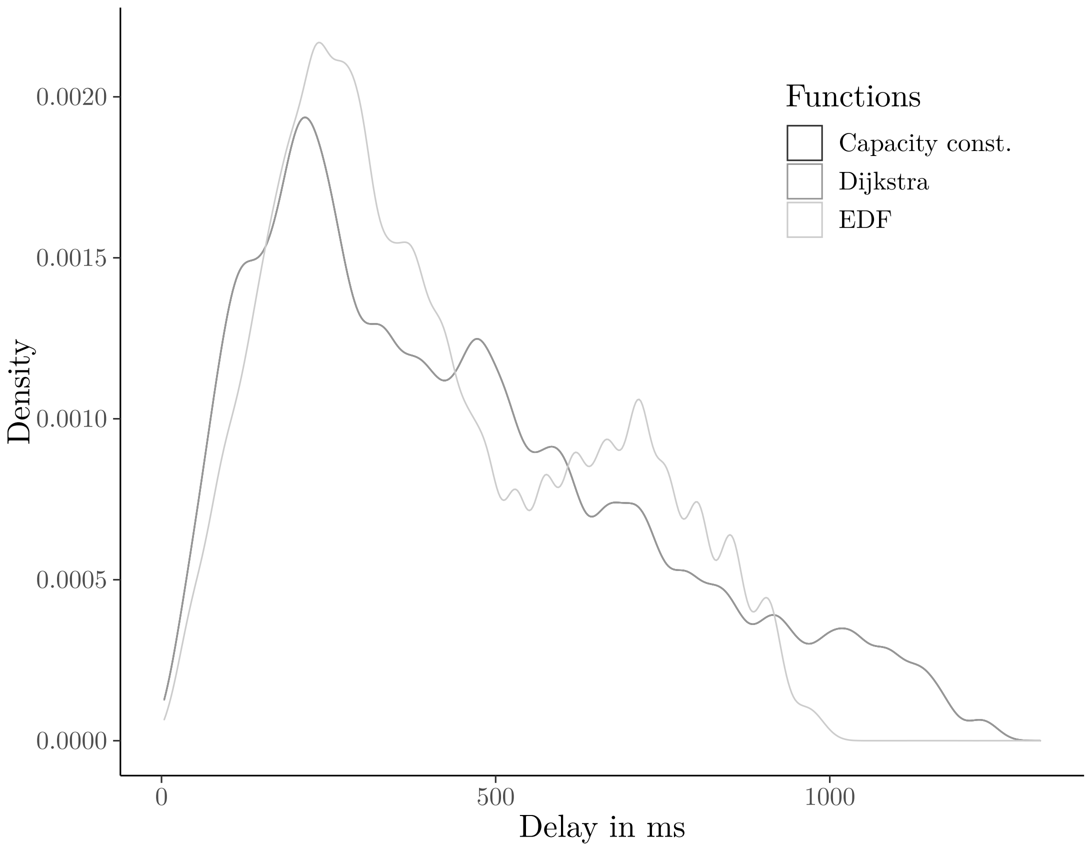


Figure 4.7: Latency distributions for the heuristics in the avionics network for $|D|$=1869. Capacity const = Dijkstra

4.3.5 Memory Usage During Calculation

While there are some differences in the memory consumption (with EDF requiring the most as all loop-free paths are being calculated), the overall memory usage by the heuristics is very low (a few MB). Hence, memory consumption is not a limiting factor with respect to selecting a heuristic algorithm that can also be deployed online.

4.3.6 Path Length

As depicted in figure 4.9, EDF gives longer paths if a more critical demand's delay bound would otherwise be violated. With an increase in the number of demands, it is more likely that a critical demand is preventing a shorter path being used. For all algorithms investigated, the secondary (backup) paths are longer than the primary (default) paths, as shown in figure 4.10.

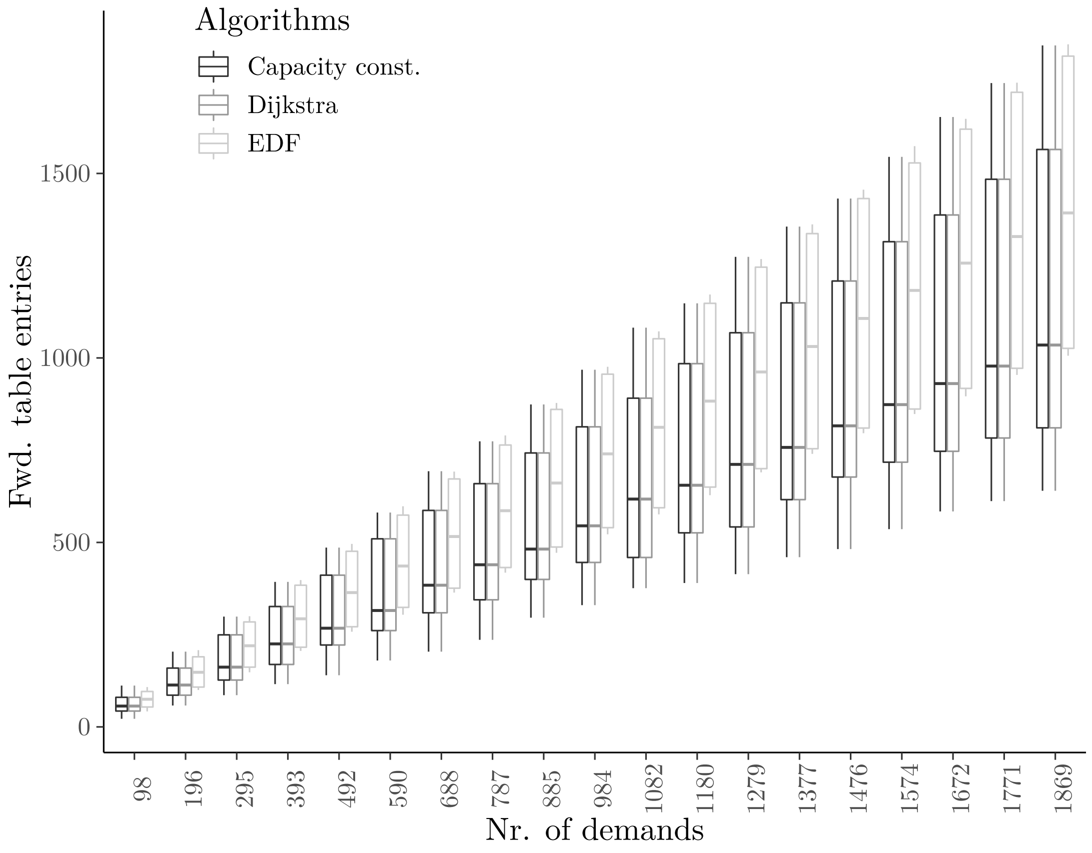


Figure 4.8: Forwarding table entries using heuristics for the avionics network

While longer paths make it more likely that a path is affected by any failure, even for EDF, the majority of demands use the direct path. Only for the backup path, longer paths are used. Consequently, EDF is only less preferential for multiple failures.

4.3.7 Violation of Constraints

In figure 4.11, the number of delay bound violations for the avionics network is shown. While the EDF heuristic yields significantly lower mean number of violations than Dijkstra's algorithm (μ=447.3 vs. μ=841.5), it is still unacceptable for safety-critical traffic that over 20% of demands have their delay bounds violated. In addition, for different topologies or demand sets, violations of infrastructure constraints may occur even if the delay bound is not as critical as for avionics. With respect to the nobel-germany network, only latency violations could be recorded for Dijkstra and Lim. capa. EDF didn't result in violations. In this case, no differences between the two objective functions has been observed. While a mean of 1.4 violations only occurred for the tightest latency vi-

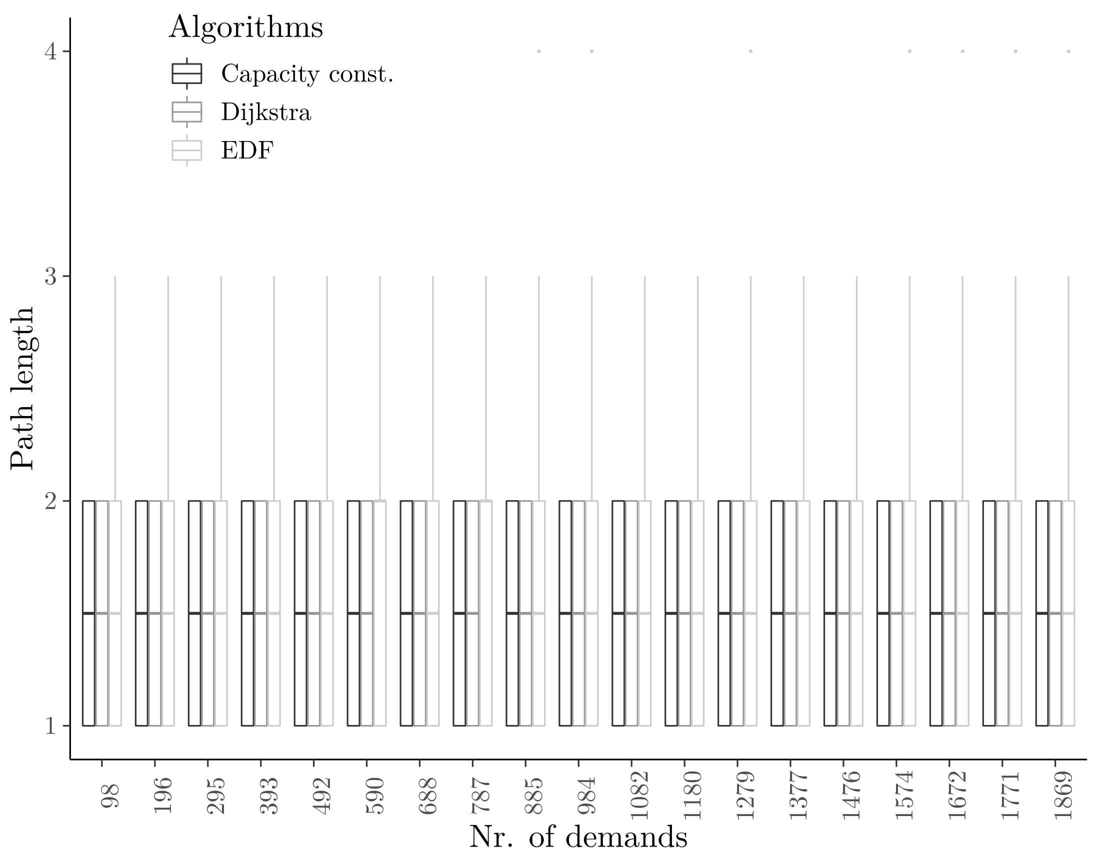


Figure 4.9: Path length using heuristics for the avionics network

olations that could still be solved, it is unacceptable for the demands of a safety-critical system to be thus violated. Yet it might be acceptable for non-safety-critical systems such as mission computers.

Moreover, it has been observed for the networks not depicted that the capacity-limited heuristic tends to result in delay bound violations, while the EDF heuristic causes the capacity limit to be exceeded. Consequently, no heuristic is able to provide configuration templates for safety-critical traffic. However, one could consider using the EDF algorithm to provide resilience to mission-critical traffic until no more traffic could be routed without violating constraints.

4.4 Optimisation

In addition to the heuristics described above, five optimisation functions have been compared using the same criteria.

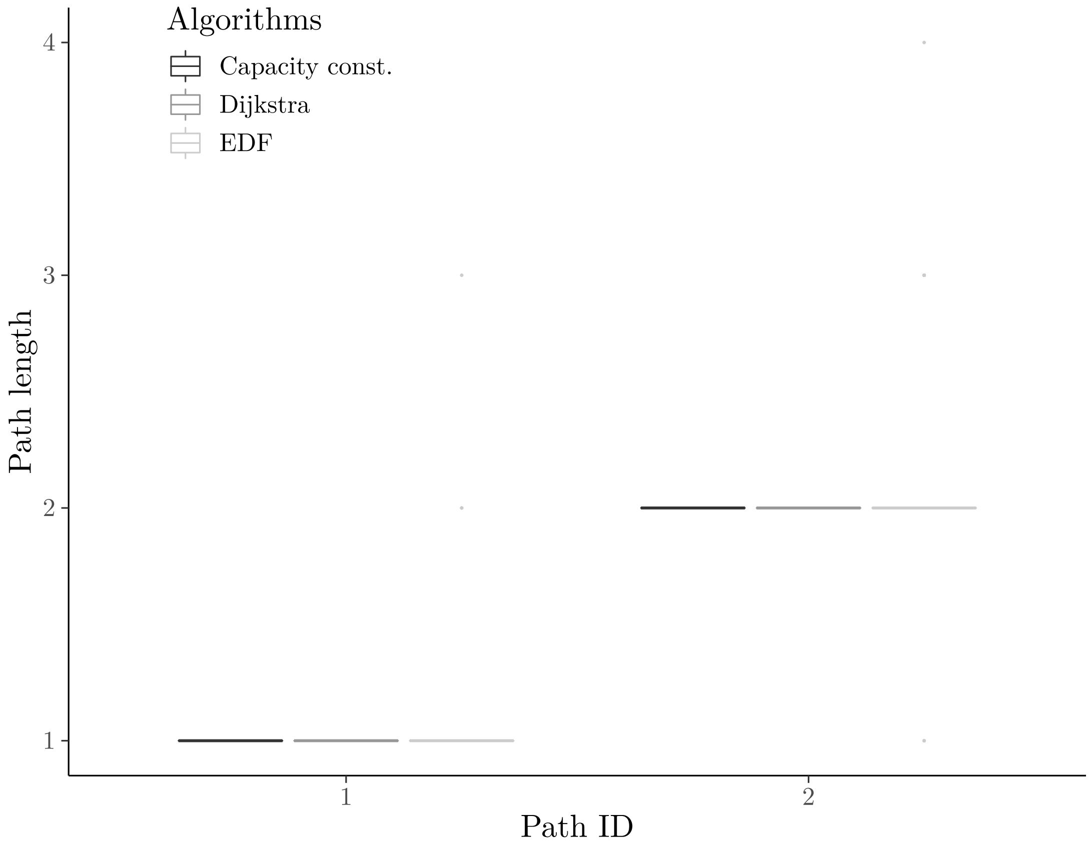


Figure 4.10: Comparison of path length and resilience using heuristics for the avionics network

Beyond those, the number of available node- and link-disjoint paths is studied here to see how many failures could be tolerated before some network traffic is lost.

The same ten problem instances as for the heuristics have been used here. Where the respective performance characteristic was not the objective function, secondary optimisations have been performed with the result of the first optimisation as a constraint. Thus where several solutions for the primary optimisation exist, the comparison is fair with respect to the selected performance goal.

The key results of this section have already been published in [143]. However, this thesis gives a more comprehensive insight into the problem by specifically analysing the effect of specific constraints on the performance. Additionally, the paper only considered some of the parameters that are detailed below.

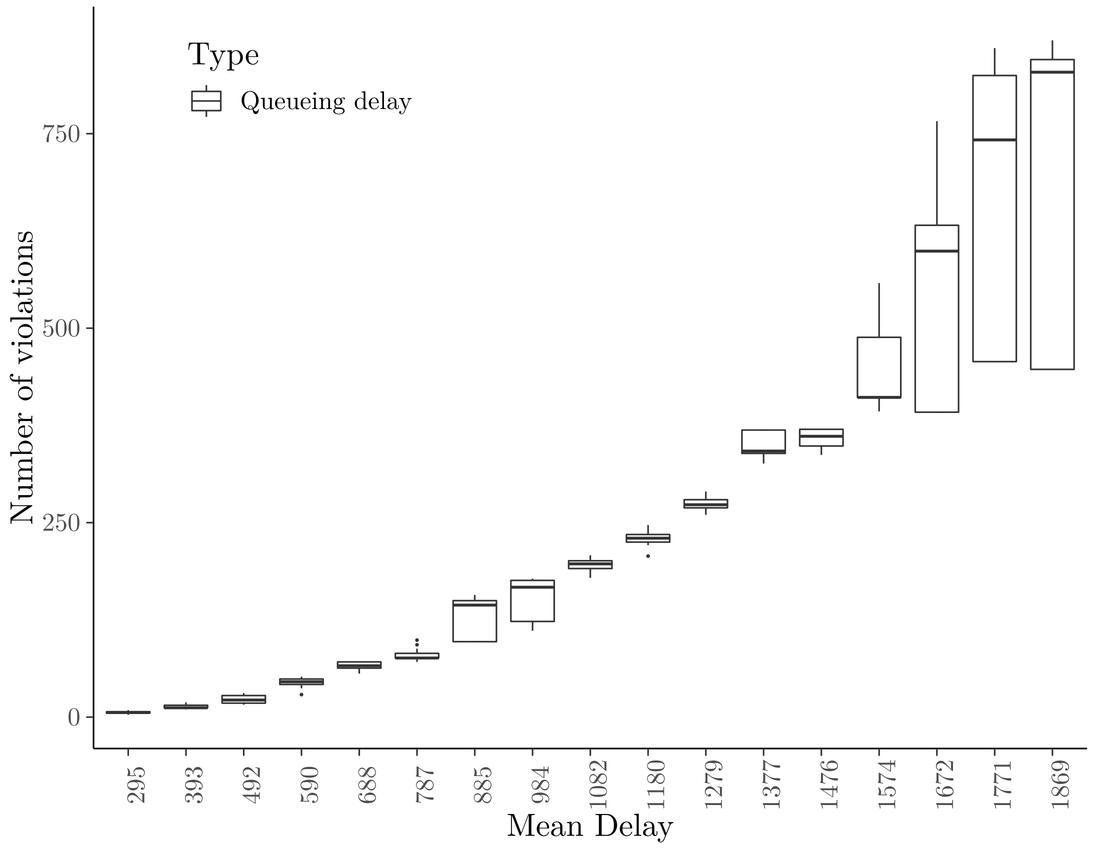


Figure 4.11: Violations of requirements using heuristics for the avionics network

4.4.1 Computing Resources

For the results given in the following paragraphs, the commercial Gurobi [67] solver has been used on the following machines:

1. Ubuntu 16.04, 48 GB of RAM and a 12-core Intel Xeon CPU W3690 @ 3.47GHz
2. CentOS 6.9, 256 GB of RAM and a 24-core Xeon CPU E5-2667 0 @ 2.90GHz

While the open-source GNU linear programming kit (GLPK)[62] has also been used initially, it was found that due to internal memory limitations that limit the number of constraints, this solver could only deal with up to about 1120 demands for the minimisations in the present formulation of the optimisation problem. A more detailed analysis of the computing resources needed is given in section 4.4.8.

The more powerful machine was used for the avionics problems, as the large demand set exceeded the available memory of the smaller machine from about 700 demands onwards.

4.4.2 Effect of Additional Constraints

As the application of SDN to critical system requires some additional constraints, the addition thereof to a standard routing problem has been studied in more detail. The effect of adding these constraints on the investigated performance characteristics is provided in table 4.6 to table 4.13 for each of the studied objective functions, here for the ATLANTA network. In these tables, the full demand set carries no extra indication, while the term *sim.* indicates a simplified demand set using just capacity limits. In addition, the following abbreviations are used: *R* indicates the resilience constraint being applied, *norm.* the usage of the normalised objective function as given in [57], while *DB* indicates limiting the delay bound, and *FTE* the forwarding table entries, respectively.

It has been observed that the resilience constraints have the most significant impact on all performance characteristics. This is to be expected since constraint doubles number of demands that need to be placed.

In contrast, including forwarding tables does not have a pronounced effect for most functions, as they generally do not operate near maximum capacity. An exception here is the *minimum number of forwarding table entries* function, since the two have opposite goals, with latency reducing the average path length and forwarding table increasing it. The latency requirement affects the *minimum number of forwarding table entries* objective function, since there are some flows that would otherwise violate the requirements.

4.4.3 Bandwidth

As mentioned above, the bandwidth required by the various routing strategies provides information on how well the network can handle additional demands. In figure 4.12, the distribution of link usage is shown. As expected, for most networks the *maximum resilience* yields the highest resource usage, since it allocates the most traffic on the network, both link and switch usage are the highest. However, for the ATLANTA network, the *minimum link usage* actually gives in the highest maximum usage, while the mean is the lowest. As only the sum of all link usages is minimised, the high usage of one link is compensated by low usage of others. Additionally, as only very high usage is penalised by this function, it does not yield good results in the low and medium usage, as it is the case for the networks studied in this work.

Table 4.6: Effect of Constraints for the ATLANTA Network – Cost

Network	Min.	Median	Mean	Max.
Lim. capa.	12576.8	12576.8	$228{\cdot}10^{9}$	$1.52{\cdot}10^{12}$
Lim. capa. (norm.)	159.767	159.767	159.767	159.767
Lim. capa. (DB)	106.055	108.321	109.333	114.797
Lim. capa. (DB,R)	11736.3	12245	$253{\cdot}10^{9}$	$1.52{\cdot}10^{12}$
Max. res.	$1.86{\cdot}10^{12}$	$1.87{\cdot}10^{12}$	$1.88{\cdot}10^{12}$	$1.92{\cdot}10^{12}$
Max. res. (DB,R)	$1.40{\cdot}10^{12}$	$1.49{\cdot}10^{12}$	$1.48{\cdot}10^{12}$	$1.53{\cdot}10^{12}$
Max. res. (FTE)	$1.69{\cdot}10^{12}$	$1.69{\cdot}10^{12}$	$1.69{\cdot}10^{12}$	$1.69{\cdot}10^{12}$
Max. res. (sim.)	$1.80{\cdot}10^{12}$	$1.80{\cdot}10^{12}$	$1.80{\cdot}10^{12}$	$1.80{\cdot}10^{12}$
Min. cost	$1.00{\cdot}10^{12}$	$1.43{\cdot}10^{12}$	$1.39{\cdot}10^{12}$	$1.491{\cdot}10^{12}$
Min. cost (DB)	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$
Min. cost(FTE)	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$
Min. cost(R)	$9.99{\cdot}10^{11}$	$9.99{\cdot}10^{11}$	$9.99{\cdot}10^{11}$	$9.99{\cdot}10^{11}$
Min. cost(sim.)	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$	$3.46{\cdot}10^{11}$
Min. fwd. tab.	$1.65{\cdot}10^{12}$	$1.72{\cdot}10^{12}$	$1.74{\cdot}10^{12}$	$1.85{\cdot}10^{12}$
Min. fwd. tab. (DB)	$5.58{\cdot}10^{11}$	$6.07{\cdot}10^{11}$	$6.60{\cdot}10^{11}$	$1.06{\cdot}10^{12}$
Min. fwd. tab. (FTE)	$5.25{\cdot}10^{11}$	$5.25{\cdot}10^{11}$	$5.25{\cdot}10^{11}$	$5.25{\cdot}10^{11}$
Min. fwd. tab. (R)	$1.47{\cdot}10^{12}$	$1.47{\cdot}10^{12}$	$1.47{\cdot}10^{12}$	$1.47{\cdot}10^{12}$
Min. fwd. tab. (sim.)	$5.25{\cdot}10^{11}$	$5.25{\cdot}10^{11}$	$5.25{\cdot}10^{11}$	$5.25{\cdot}10^{11}$
Min. delay	$1.43{\cdot}10^{12}$	$1.53{\cdot}10^{12}$	$1.53{\cdot}10^{12}$	$1.65{\cdot}10^{12}$
Min. delay (FTE)	$5.28{\cdot}10^{11}$	$5.83{\cdot}10^{11}$	$6.01{\cdot}10^{11}$	$6.97{\cdot}10^{11}$
Min. delay (sim.)	$5.00{\cdot}10^{11}$	$5.56{\cdot}10^{11}$	$5.52{\cdot}10^{11}$	$5.85{\cdot}10^{11}$

In contrast, for the avionics network, the *minimum cost* results in much lower usage, as the link usage is directly linked with the cost for this network. Furthermore, *minimum queueing delay* and *minimum forwarding table entries* objective functions also give lower usage, since both algorithms prefer short paths (see below), which in turn leads to an even distribution across the network and hence to an overall lower link usage.

Similar results can be observed for the bandwidth required at the switch, as depicted in figure 4.13. For the DFN network, even the most demanding *maximum resilience* uses at most half of the available capacity. This objective function experiences a high variance that can be explained by an imbalanced demand set. There is a significant difference between switches that are connected to sources of many traffic demands and hence have to handle significantly more traffic than those that are mainly forwarding traffic.

The *minimum latency* objective displays a higher minimum and mean usage with less variance between the different switches. This is because this algorithm ensures that queueing delay is minimised for all demands, consequently it is aiming to distribute

Table 4.7: Effect of Constraints for the ATLANTA Network – Time

Network	**Min.**	**Median**	**Mean**	**Max.**
Lim. capa.	1.318	1.333	80.11	526.532
Lim. capa. (norm.)	1.263	1.269	1.282	1.399
Lim. capa. (DB)	18.288	19.321	19.629	21.17
Lim. capa. (DB,R)	248.692	322.572	399.517	900.511
Max. res.	301.4	361.6	416.68	608.2
Max. res. (FTE)	7.1	10.35	9.87	13.3
Max. res. (sim.)	2.8	2.9	2.92	3.1
Min. cost	468.5	537.6	581.525	726.2
Min. cost (DB)	30.4	39.7	40.011	47.4
Min. cost(FTE)	0.3	0.3	0.32	0.4
Min. cost(R)	4.7	5.15	5.74	7.3
Min. cost(sim.)	0.3	0.3	0.3	0.3
Min. fwd. tab.	312.1	327.35	364.25	551
Min. fwd. tab. (DB)	8313.8	14726.9	14669.67	18691.2
Min. fwd. tab. (FTE)	0.2	0.2	0.2	0.2
Min. fwd. tab. (R)	3.5	5.3	4.78	6
Min. fwd. tab. (sim.)	0.1	0.1	0.1	0.1
Min. delay	217.5	253.7	265.45	321.1
Min. delay (R)	1317.9	1318.05	1318.11	1318.4
Min. delay (sim.)	12.6	14.8	15.19	20

traffic as evenly as possible across the network. This strategy is also reflected in the lower bandwidth requirement of the links as shown above.

The slight differences between the different runs of the *minimum number of forwarding table entries* for the Deutsches Forschungsnetz (DFN) network is most probably related to specific values of the individual demand sets. Only the DFN network shows a significant variation between sets. Other than that, this objective provides the lowest maximum usage for this aspect, with a slightly elevated minimum usage compared to other minimisations objectives.

With respect to resource utilisation, no variation could be observed with varying the delay bound. Consequently, tighter delay bounds do not have an effect of the distribution of link utilisation.

Generally, all optimisation functions are suitable to generate configuration templates with respect to the bandwidth used, as all are below the maximum limit. Notwithstanding, in those cases where only one failure needs to be tolerated, the *minimum cost* provides the lowest resource usage. Yet in most critical systems, it is not sufficient to tolerate only one failure. Here, the *maximum resilience* objective function is more

Table 4.8: Effect of Constraints for the ATLANTA Network – Used Bandwidth (Link)

Network	**Min.**	**Median**	**Mean**	**Max.**
Lim. capa.	0	$19.09{\cdot}10^3$	$21.5{\cdot}10^3$	$43.65{\cdot}10^3$
Lim. capa. (norm.)	$4.286{\cdot}10^3$	$19.69{\cdot}10^3$	$21.29{\cdot}10^3$	$43.53{\cdot}10^3$
Lim. capa. (DB)	0	$6.58{\cdot}10^3$	$7.82{\cdot}10^3$	$26.56{\cdot}10^3$
Lim. capa. (DB,R)	0	$24.87{\cdot}10^3$	$23.24{\cdot}10^3$	$43.65{\cdot}10^3$
Max. res.	$4.47{\cdot}10^3$	$32.39{\cdot}10^3$	$29.88{\cdot}10^3$	$40{\cdot}10^3$
Max. res. (DB,R)	$5.303{\cdot}10^3$	$25.38{\cdot}10^3$	$23.82{\cdot}10^3$	$39.94{\cdot}10^3$
Max. res. (FTE)	$7.772{\cdot}10^3$	$29{\cdot}10^3$	$26.78{\cdot}10^3$	$40{\cdot}10^3$
Max. res. (sim.)	$4.134{\cdot}10^3$	$35.68{\cdot}10^3$	$28.79{\cdot}10^3$	$40{\cdot}10^3$
Min. cost	$4.134{\cdot}10^3$	$22.12{\cdot}10^3$	$22.79{\cdot}10^3$	$40{\cdot}10^3$
Min. cost (DB)	$1.057{\cdot}10^3$	$4.287{\cdot}10^3$	$6.376{\cdot}10^3$	$23.86{\cdot}10^3$
Min. cost(FTE)	$1.057{\cdot}10^3$	$4.287{\cdot}10^3$	$6.376{\cdot}10^3$	$23.86{\cdot}10^3$
Min. cost(R)	$4.134{\cdot}10^3$	$13.69{\cdot}10^3$	$17.73{\cdot}10^3$	$40{\cdot}10^3$
Min. cost(sim.)	$1.057{\cdot}10^3$	$4.287{\cdot}10^3$	$6.376{\cdot}10^3$	$23.86{\cdot}10^3$
Min. fwd. tab.	$5.142{\cdot}10^3$	$27.84{\cdot}10^3$	$26.72{\cdot}10^3$	$40{\cdot}10^3$
Min. fwd. tab. (DB)	$298{\cdot}10^0$	$8.646{\cdot}10^3$	$10.47{\cdot}10^3$	$39.12{\cdot}10^3$
Min. fwd. tab. (FTE)	$252{\cdot}10^0$	$7.287{\cdot}10^3$	$8.297{\cdot}10^3$	$24.96{\cdot}10^3$
Min. fwd. tab. (R)	$5.426{\cdot}10^3$	$27.26{\cdot}10^3$	$23.06{\cdot}10^3$	$40{\cdot}10^3$
Min. fwd. tab. (sim.)	$252{\cdot}10^0$	$7.287{\cdot}10^3$	$8.297{\cdot}10^3$	$24.96{\cdot}10^3$
Min. delay	$4.811{\cdot}10^3$	$24.78{\cdot}10^3$	$24.61{\cdot}10^3$	$40{\cdot}10^3$
Min. delay (FTE)	$338{\cdot}10^0$	$8{\cdot}10^3$	$10.21{\cdot}10^3$	$39.82{\cdot}10^3$
Min. delay (sim.)	$78.52{\cdot}10^0$	$8.267{\cdot}10^3$	$9.357{\cdot}10^3$	$33.13{\cdot}10^3$

suitable, as it provides the maximum number of failure tolerance, while still accepting the capacity constraints.

4.4.4 Calculation Time

To study whether runtime optimisation is possible, it is necessary to study calculation times. As the satisfaction of all constraints is guaranteed with optimisation, there is no prior reason that these cannot be performed during operation. Here, the calculation was aborted when the gap between the current solution and the optimal solution was less than 1 percent.

From figure 4.14, it can be observed that due to the larger number of free variables and the interdependence between them, the *minimum queueing delay* and *maximum resilience* have the largest difference between mean and maximum calculation time. While calculation times for *minimum latency* are all within the same order of magnitude, the *max. resilience* and *minimum forwarding table entries* require – on average – up to 100 times the calculation time of the *minimum cost* objective function. It should be

Table 4.9: Effect of Constraints for the ATLANTA Network – Used Bandwidth (SW)

Network	**Min.**	**Median**	**Mean**	**Max.**
Lim. capa.	$27.38{\cdot}10^3$	$123.5{\cdot}10^3$	$126.1{\cdot}10^3$	$219.7{\cdot}10^3$
Lim. capa. (norm.)	$46.06{\cdot}10^3$	$120.5{\cdot}10^3$	$124.9{\cdot}10^3$	$217{\cdot}10^3$
Lim. capa. (DB)	$8.582{\cdot}10^3$	$44.79{\cdot}10^3$	$45.88{\cdot}10^3$	$104.4{\cdot}10^3$
Lim. capa. (DB,R)	$25.7{\cdot}10^3$	$147.5{\cdot}10^3$	$136.4{\cdot}10^3$	$237.2{\cdot}10^3$
Max. res.	$26.07{\cdot}10^3$	$178.9{\cdot}10^3$	$175.3{\cdot}10^3$	$316.9{\cdot}10^3$
Max. res. (DB,R)	$34.65{\cdot}10^3$	$137.1{\cdot}10^3$	$139.7{\cdot}10^3$	$261.3{\cdot}10^3$
Max. res. (FTE)	$35.02{\cdot}10^3$	$145.8{\cdot}10^3$	$157.1{\cdot}10^3$	$263.3{\cdot}10^3$
Max. res. (sim.)	$16.98{\cdot}10^3$	$177.2{\cdot}10^3$	$168.9{\cdot}10^3$	$316.9{\cdot}10^3$
Min. cost	$16.98{\cdot}10^3$	$128.1{\cdot}10^3$	$133.7{\cdot}10^3$	$271.6{\cdot}10^3$
Min. cost (DB)	$7.805{\cdot}10^3$	$24.57{\cdot}10^3$	$37.41{\cdot}10^3$	$103.7{\cdot}10^3$
Min. cost(FTE)	$7.805{\cdot}10^3$	$24.57{\cdot}10^3$	$37.41{\cdot}10^3$	$103.7{\cdot}10^3$
Min. cost(R)	$16.98{\cdot}10^3$	$82.57{\cdot}10^3$	$104{\cdot}10^3$	$212.9{\cdot}10^3$
Min. cost(sim.)	$7.805{\cdot}10^3$	$24.57{\cdot}10^3$	$37.41{\cdot}10^3$	$103.7{\cdot}10^3$
Min. fwd. tab.	$31.11{\cdot}10^3$	$165.4{\cdot}10^3$	$156.8{\cdot}10^3$	$257.4{\cdot}10^3$
Min. fwd. tab. (DB)	$9.725{\cdot}10^3$	$55.92{\cdot}10^3$	$61.45{\cdot}10^3$	$160{\cdot}10^3$
Min. fwd. tab. (FTE)	$12.65{\cdot}10^3$	$45.74{\cdot}10^3$	$48.68{\cdot}10^3$	$128.2{\cdot}10^3$
Min. fwd. tab. (R)	$26.37{\cdot}10^3$	$146.1{\cdot}10^3$	$135.3{\cdot}10^3$	$236.6{\cdot}10^3$
Min. fwd. tab. (sim.)	$12.65{\cdot}10^3$	$45.74{\cdot}10^3$	$48.68{\cdot}10^3$	$128.2{\cdot}10^3$
Min. delay	$26.24{\cdot}10^3$	$142.1{\cdot}10^3$	$144.4{\cdot}10^3$	$282.3{\cdot}10^3$
Min. delay (FTE)	$8.163{\cdot}10^3$	$57.28{\cdot}10^3$	$59.92{\cdot}10^3$	$135{\cdot}10^3$
Min. delay (sim.)	$9.196{\cdot}10^3$	$52.46{\cdot}10^3$	$54.9{\cdot}10^3$	$143.9{\cdot}10^3$

noted that this figure has been zoomed in to only show the mean and quartile. The complete results can be found in the appendix (C.2).

Unlike with the resource usage, tightening of the delay bound significantly increases calculation times for the networks investigated. Here, the most significant impact can be observed on the *maximum resilience* objective function. As expected, the calculation time increases with the number of demands, for the *minimum cost*, the median calculation time increases by a factor of 13 between $n = 98$ and $n = 1869$. Yet the largest increase can again be observed for the *maximum resilience* objective function, where the median increases from 72.92 seconds to $3.044{\cdot}10^3$ seconds.

As the calculation times are significant even for average cases, performing online optimisations only upon the occurrence of a failure is not a feasible strategy for safety-critical systems. While feasible solutions that satisfy all constraints but are not optimal may be obtained more quickly, the lack of suitable computing resources (especially for networks on vehicles such as cars and aircraft) make it improbable that those calculations will be performed during system operation. Beyond that, current certification requirements do not allow for non-deterministic system behaviour, which is problematic as there may

Table 4.10: Effect of Constraints for the ATLANTA Network – Delay

Network	**Min.**	**Median**	**Mean**	**Max.**
Lim. capa.	14	165	177.1	463
Lim. capa. (norm.)	14	149	158.271	435
Lim. capa. (DB)	3	56	63.721	190
Lim. capa. (DB,R)	16	159	173.461	427
Max. res.	0.174	138.529	172.065	724.981
Max. res. (DB,R)	17	121	143.774	541
Max. res. (FTE)	24	160	169.805	441
Max. res. (sim.)	15	157.06	183.246	651.781
Min. cost	7	165.338	193.775	654.793
Min. cost (DB)	2	40	42.133	99
Min. cost(FTE)2	40	42.133	99	
Min. cost(R)	7.513	128.244	132.188	283.348
Min. fwd. tab.	6.958	263.603	276.407	688.306
Min. fwd. tab. (DB)	2	83	103.486	579
Min. fwd. tab. (FTE)	4	66.5	75.919	182
Min. fwd. tab. (R)	7	213.946	247.174	689.368
Min. fwd. tab. (sim.)	4	66.5	75.919	182
Min. delay	15.048	206.564	211.666	593.081
Min. delay (FTE)	0	1	77.644	746

be more than one optimal solution to any given problem, and solvers would need to be certified to produce deterministic results.

With respect to calculation time, the *minimum link utilisation* is the most predictable, as it does not produce significant outliers, albeit with a slightly higher mean. Consequently, it is the most suitable optimisation to generate templates in cases where very frequent updates to both the topology and the demand set occur. However, in more static safety-critical systems, the increased calculation time of the *maximum resilience* may be justified by the additional backup paths it provides.

4.4.5 Cost

In figure 4.15, a comparison with respect to cost is shown. For the avionics network, the *minimum link usage* calculated the cost as done in literature. As all links are far from being fully utilised, it is lowest for this case. As expected, the *minimum cost* yields the lowest number and *maximum resilience* the highest amongst the other objective functions. While the remaining functions are within the same order of magnitude, the minimum cost using the *minimum queueing delay* is nearly 27% lower than that of the *minimum link usage*.

Table 4.11: Effect of Constraints for the ATLANTA Network – Path Length

Network	**Min.**	**Median**	**Mean**	**Max.**
Lim. capa.	1	4	4.016	11
Lim. capa. (norm.)	1	4	3.886	10
Lim. capa. (DB)	1	3	3.202	9
Lim. capa. (DB,R)	1	4	3.926	10
Max. res.	1	101	101.972	209
Max. res. (DB,R)	1	105	105.794	209
Max. res. (FTE)	1	105	105.917	209
Max. res. (sim.)	1	98	99.471	209
Min. cost	1	3	2.669	7
Min. cost (DB)	1	3	2.571	5
Min. cost(FTE)	1	3	2.571	5
Min. cost(R)	1	3	3.114	9
Min. cost(sim.)	1	3	2.571	5
Min. fwd. tab.	3	12	10.561	19
Min. fwd. tab. (DB)	1	3	3.975	15
Min. fwd. tab. (FTE)	1	3	3.338	8
Min. fwd. tab. (R)	1	4	4.79	15
Min. fwd. tab. (sim.)	1	3	3.338	8
Min. delay	1	11.5	10.949	22
Min. delay (FTE)	1	4	4.36	22
Min. delay (sim.)	1	7	7.229	19

For the other networks, the link costs from `SND-LIB` are used. While the objective of *minimum link usage* is also to reduce cost, only link usage close to the capacity limit is significantly penalised. Thus, the resulting cost for this function is between 25%-50% higher than for *minimum cost*, if the network does not operate close to capacity, as it is the case for the DFN network.

Regarding the *maximum resilience*, the increasingly smaller delay bound affects the cost by reducing the number of resilient paths, which in turn reduces the resulting cost. Consequently, there is little difference to the other objective functions in those networks wherever the number of resilient paths is close to two.

In contrast, the aim to reduce the number of flows per link in order to minimise queueing delay results in more expensive links to be used by the *minimum latency* function, which in turn raises the cost.

With respect to the effect of the latency bound, it can be observed that tighter latency bounds result in higher cost. Yet the values vary between the networks and functions.

Table 4.12: Effect of Constraints for the ATLANTA Network – Forwarding Table Entries

Network	Min.	Median	Mean	Max.
Lim. capa.	43	119	112.44	179
Lim. capa. (norm.)	60	110	108.8	173
Lim. capa. (DB)	18	39	44.833	78
Lim. capa. (DB,R)	38	117	109.933	172
Max. res.	83	282.5	263.133	453
Max. res. (DB,R)	39	122	116.293	190
Max. res. (FTE)	47	148	131.6	190
Max. res. (sim.)	30	146	135.533	219
Min. cost	30	221	212.022	368
Min. cost (DB)	14	32	36	72
Min. cost(FTE)	14	32	36	72
Min. cost(R)	30	83	94.467	176
Min. cost(sim.)	14	32	36	72
Min. fwd. tab.	57	335	293.693	436
Min. fwd. tab. (DB)	14	58	55.653	138
Min. fwd. tab. (FTE)	14	48	46.733	87
Min. fwd. tab. (R)	28	167	136.933	198
Min. fwd. tab. (sim.)	14	48	46.733	87
Min. delay	89	294.5	302.187	516
Min. delay (FTE)	16	61	61.047	159
Min. delay (sim.)	27	104	101.2	183

For the Nobel-Germany network, the difference is about 0.3% for the *minimum cost* function, but 2.4% for the *minimum queueing delay.*

With respect to the resulting cost, the selection of the most suitable objective function to generate configuration templates depends on the definition. As the *minimum link utilisation* function calculates cost based on link utilisation, it is only efficient if this is significant. In contrast, if the true link costs as per definition are used, the *minimum cost* is better suited. Obviously, using more paths is more expensive as well, thus the *maximum resilience* will result in the most expensive templates. Notwithstanding, for safety-critical system, this increased cost may be justified by the additional resilience

Table 4.13: Effect of Constraints for the ATLANTA Network – Resilience

Network	Min.	Median	Mean	Max.
Max. res.	2	2	2.338	3
Max. res. (DB,R)	1	2	2.133	3
Max. res. (FTE)	1	2	2.133	3
Max. res. (sim.)	2	2	2.338	3

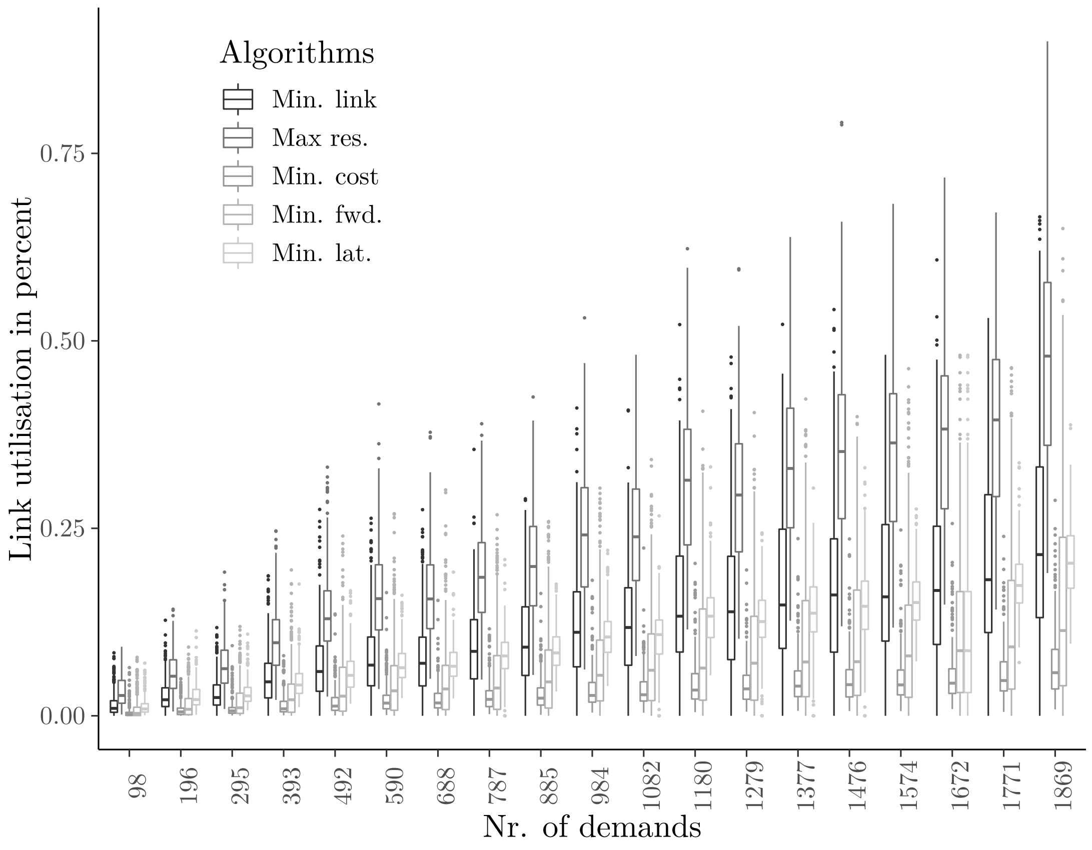


Figure 4.12: Link usage using optimisation for the avionics network

provided. In addition, this calculation does not take into account potential savings by reducing or avoiding other means of resilience, such as additional hard-wired paths.

4.4.6 Delay

Transmission, propagation and processing delay do not vary significantly between objective functions as they depend on hardware or topology characteristics. Consequently, these delay components are identical for all objective functions and hence of limited interest for the comparison of the different objective functions. Thus the analysis in the following paragraphs is for the average queueing delay that the routing of the demand yields. Consequently, the delay experienced by the demands in the avionics network can be simplified to

$$\Delta t[r,d] = \Delta t_q[r,d] = \sum_{e(i,j)\in P_{r,d}} \sum r \in R \sum d \in D \frac{x[r,d,i,j]\cdot \mathrm{bw}_d}{c_{\max}(i)}. \tag{4.2}$$

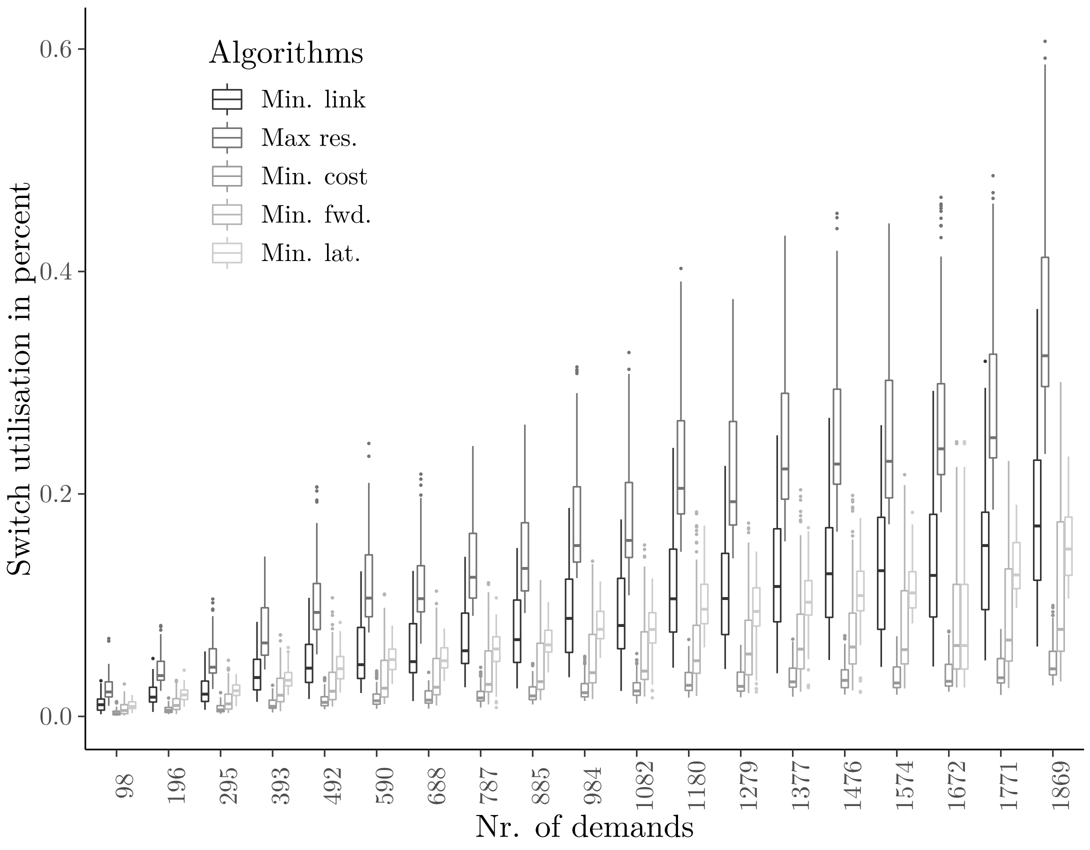


Figure 4.13: Switch usage using optimisation for the avionics network

As the latency requirements for the remaining networks are only depending on the number of demands, the bandwidth is not considered here, which gives

$$\Delta t[r,d] = \Delta t_q[r,d] = \sum_{e(i,j)\in P_{r,d}} \sum_{r\in R} \sum_{d\in D} x[r,d,i,j]. \tag{4.3}$$

The resulting distribution of queueing delay is depicted in figure 4.16. Here, the *minimum link usage* yields the highest average delay for the minimising functions where $k_d = 2$. This higher average delay is caused by the fact that the objective function only aims to reduce very crowded links operating near their maximal capacity. As this is not the case in the networks presented here, the average delay is higher. Notwithstanding, it should be noted that all constraints with respect to queueing delay are satisfied.

On the other hand, *minimum cost*, *minimum queueing delay* and *minimum forwarding table entries* cause demands to be more evenly distributed across the network, resulting

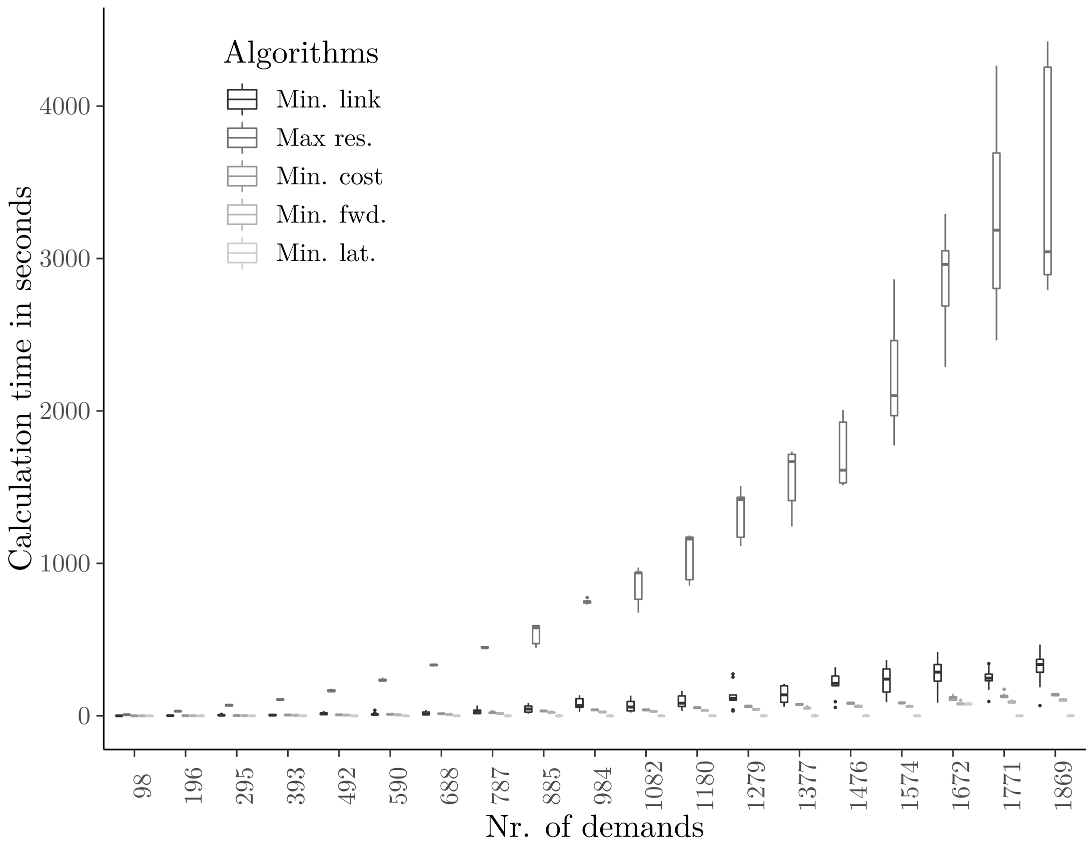


Figure 4.14: Calculation time of optimisation algorithms for the avionics network

in a smaller delay for the demands. However, it could be observed that the *minimum cost* yields a significantly lower maximum queueing delay than the *minimum queueing delay*. This is because the former explicitly considers the amount of traffic to be transported, while the latter minimises the number of demands across the network.

Obviously the *minimum queueing delay* is the most efficient for generating templates with low queueing delay. Yet the *minimum cost* optimisation also results in low queueing delay, thus making it equally suitable while also performing well in other aspects. This can be shown in further detail by comparing the density distribution as given in figure 4.17. Of the optimisation functions, the *minimum cost* the highest density of very short paths, as it punishes link usage even more severely than the *minimum latency* function. However, the density of the former rises again for very high delays above 600 ms, while the density of the other functions continuously decreases with increasing delay. The flattest curve is displayed by the *maximum resilience*, where the peak of

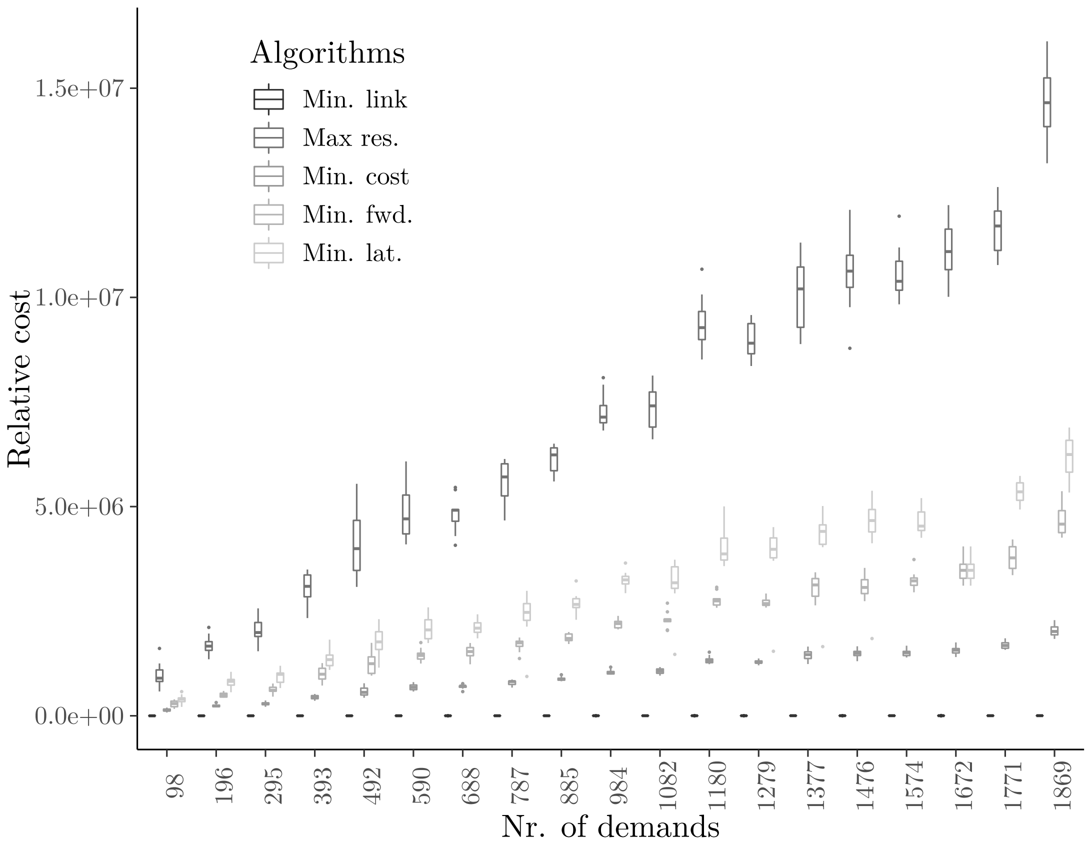


Figure 4.15: Relative cost using optimisation for the avionics network

density is also several hundred milliseconds later. However, a significantly higher traffic volume is transported here.

4.4.7 Forwarding Table Entries

In figure 4.18, the distribution of the number of forwarding table entries is shown per switch for the avionics network. Here, the *maximum resilience* yields the lowest value, despite having to route significantly more traffic. However, as no resources are shared between resilient paths, fewer forwarding rules need to be installed on every switch. Notwithstanding, the *minimum link utilisation* gives the highest value.

The *maximum resilience* is also the only function where an impact of varying latency constraints can be shown. For DFN, tighter latency constraints limit the number of node- and link-disjoint paths, consequently also reducing the amount of forwarding rules necessary. However, this effect could not be observed in all investigated networks. With tightening latency constraints, σ also increases with some significant outliers at

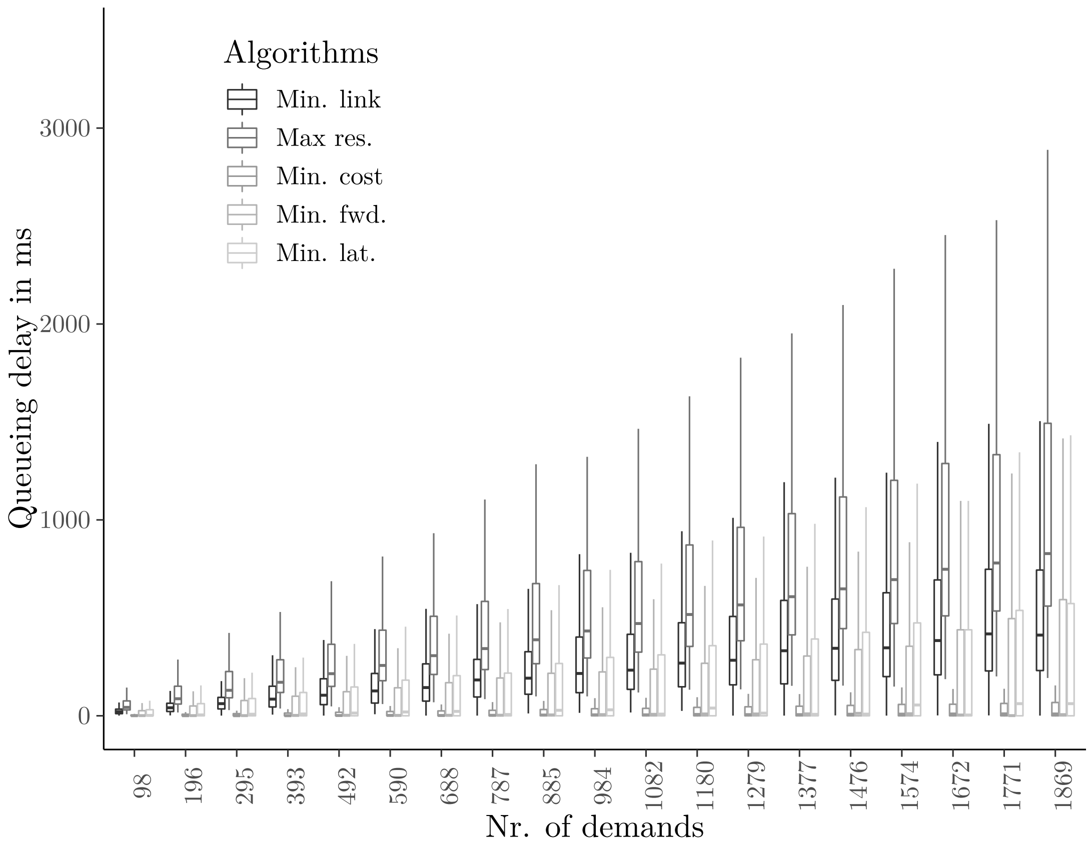


Figure 4.16: Queuing delay using optimisation for the avionics network, outliers hidden for clarity

the lower end, as the demands with the tightest latency constraints limit the number of demands placed on their link and hence also the number of forwarding rules. Yet even for the maximum number of resilient paths, less than 5% of the available TCAM memory is used.

With respect to creating configuration templates that are efficient with respect to reducing TCAM memory usage and thus energy consumption and heat dissipation, *minimum cost* and *minimum forwarding table entries*. However, despite having to route significantly more traffic, the *maximum resilience* is also efficient while providing more resilience to the network.

4.4.8 Memory Usage

In figure 4.19, the amount of random access memory (RAM) that was required to solve the *maximum resilience* problem is depicted. The memory usage has been measured using `time -v`. In this context, the avionics network has been selected since it has the

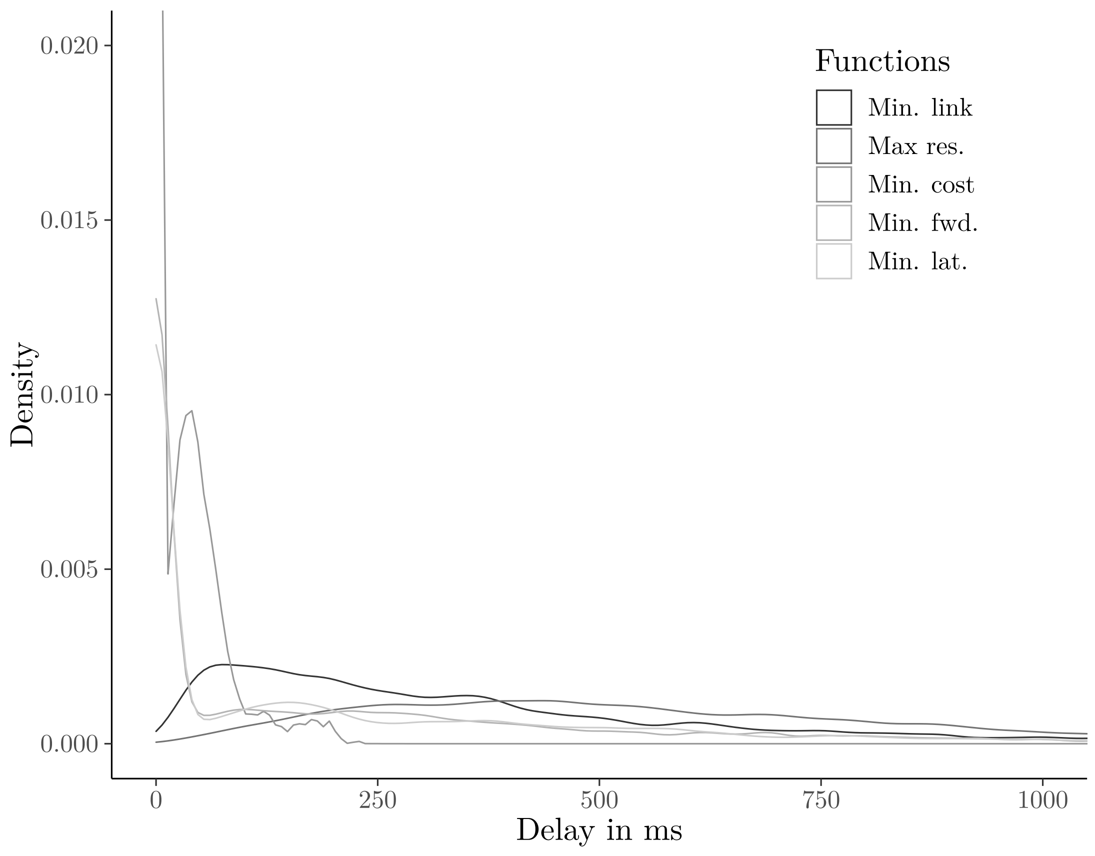


Figure 4.17: Latency distributions for the optimisation functions for the avionics network for $|D| = 1869$, cropped for clarity

most demands and the increasing demand set also allows to better assess the scalability of the problem. The smallest demand sets (98 demands) require very little memory, but the full demand set requires nearly 120 GB of RAM to hold all constraints and variables. This necessitates high-performance hardware, that will not be available during operation for safety-critical systems. Consequently, the required calculations need to be performed in advance and the results provided to the SDN controller to reconfigure the network as needed if a failure is detected.

All other optimisation functions require about the same amount of memory. Consequently, where calculation resources are limited, they are all equally suitable.

4.4.9 Path Length

Results with respect to path length are provided in figure 4.20. This figure shows that the optimisation for *minimum link usage* results in paths that are almost as long as

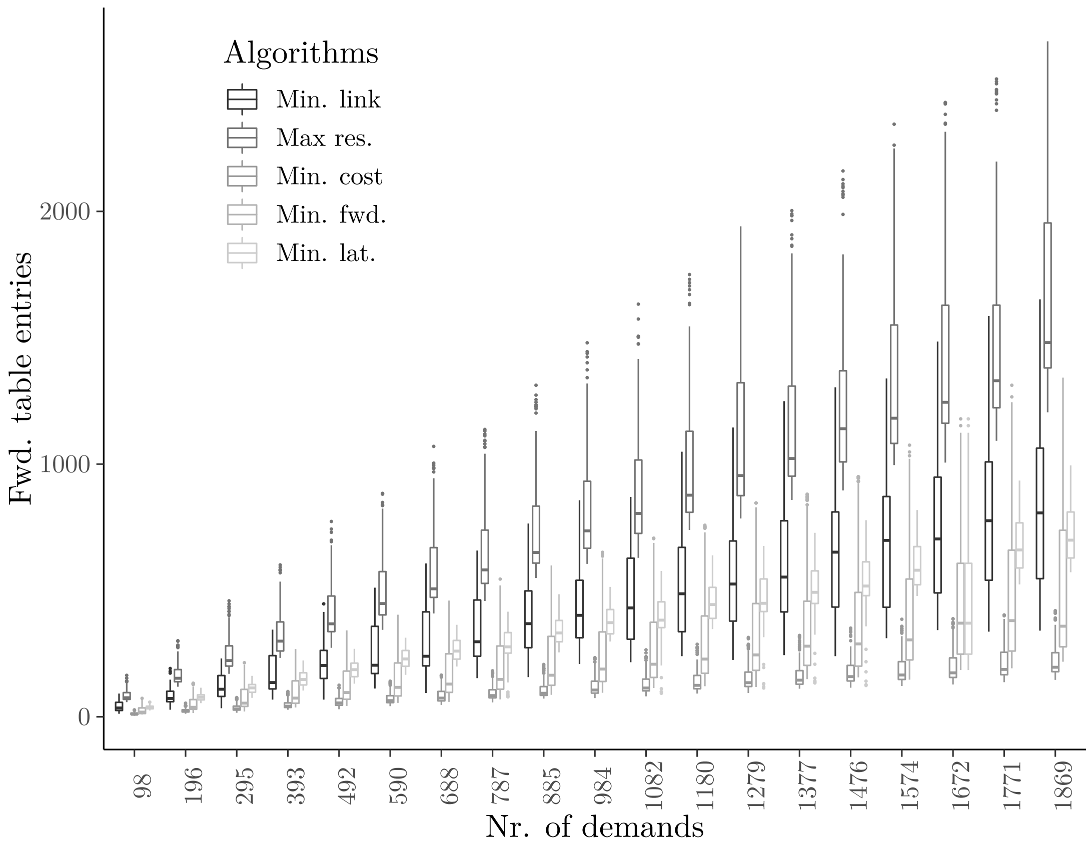


Figure 4.18: Forwarding table entries using optimisation for the avionics network

for the *maximum resilience* function, while routing less traffic. The effort to reduce link usage by evenly distributing traffic across the network results in long paths, even if the maximum capacity limit is not being reached. However, this behaviour makes it more likely that a demand is affected by a failure thus reducing resilience and is hence undesirable for safety-critical systems. Similarly, the *minimum forwarding table entries* objective function also produces longer paths. By reducing the number of rules on the switches, longer paths are encouraged. Notwithstanding, the median is two for all objective functions investigated. On the other hand, both *minimum queueing delay* and *minimum cost* functions result in comparatively short paths. This is because longer paths results in more delay for the individual flows which is not encouraged by the *minimum queueing delay* function. Along similar lines, using several links is generally more expensive than using just one as far as the *minimum cost* function is concerned. Moreover, unlike for the heuristics, there is no significant difference with respect to path length between primary and backup path for the minimisation, as depicted in

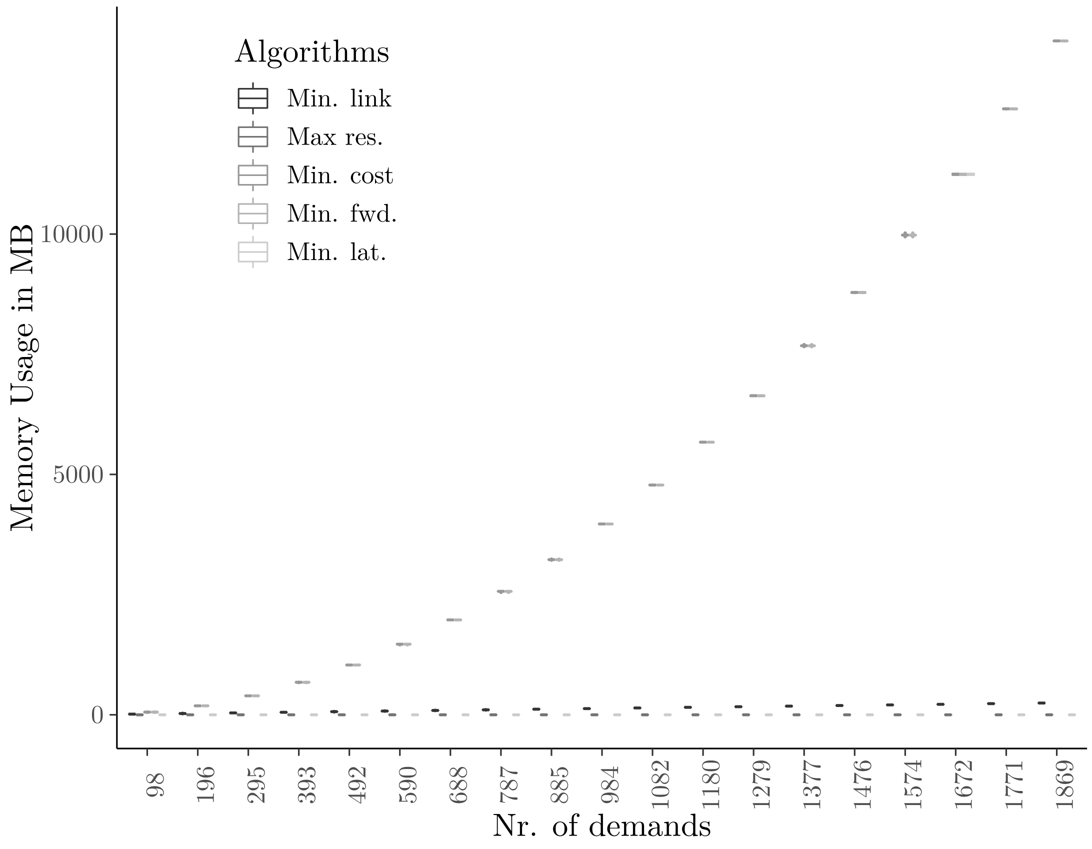


Figure 4.19: Required RAM to solve the maximum resilience optimisation problem for the avionics network

figure 4.20. Consequently, when trying to obtain configuration templates, all functions are equally suitable.

4.4.10 Maximum Resilience

Table 4.14: Distribution of Maximum Resilience

Network	**Min**	**Mean**	**Max**
ATLANTA	2	2.3	3
DFN	3	8.9	9
NOBEL-GERMANY	2	2.4	4

In table 4.14, the maximum number of node- and link-disjoint paths obtainable in the respective networks is shown. From this table, it is evident that network topology has the most significant impact on the number of available paths.

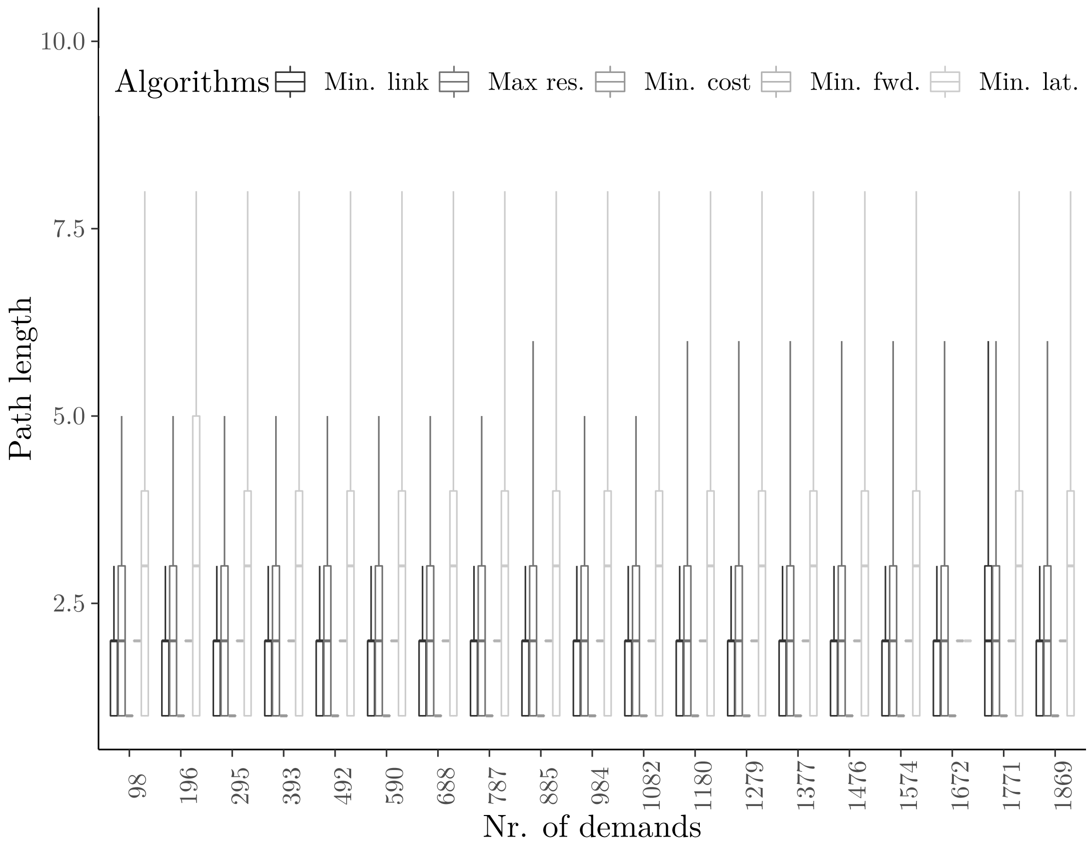


Figure 4.20: Path length using optimisation for the avionics network, outliers hidden for clarity

Yet this number is not indicative for how many failures can be tolerated before traffic is lost, since any one path may use more than one node and/or link. Nevertheless, it is essential for safety-critical communication that no common cause of failure affects resilient paths. Otherwise, several alternative network configurations could be turned unusable by a single failure.

Iterating through all possible failure cases is also unrealistic, especially when more than one failure is affecting the network, which is especially relevant for security-related incidents.

Yet surprisingly, the number of demands does not have a significant effect on the maximum resilience that can be provided, as shown in table 4.15, at least not for the avionics network. This is most likely because the demand data set already incorporates redundancy. Consequently, the paths can all be used simultaneously without violating constraints, especially considering that only little traffic needs to be transported.

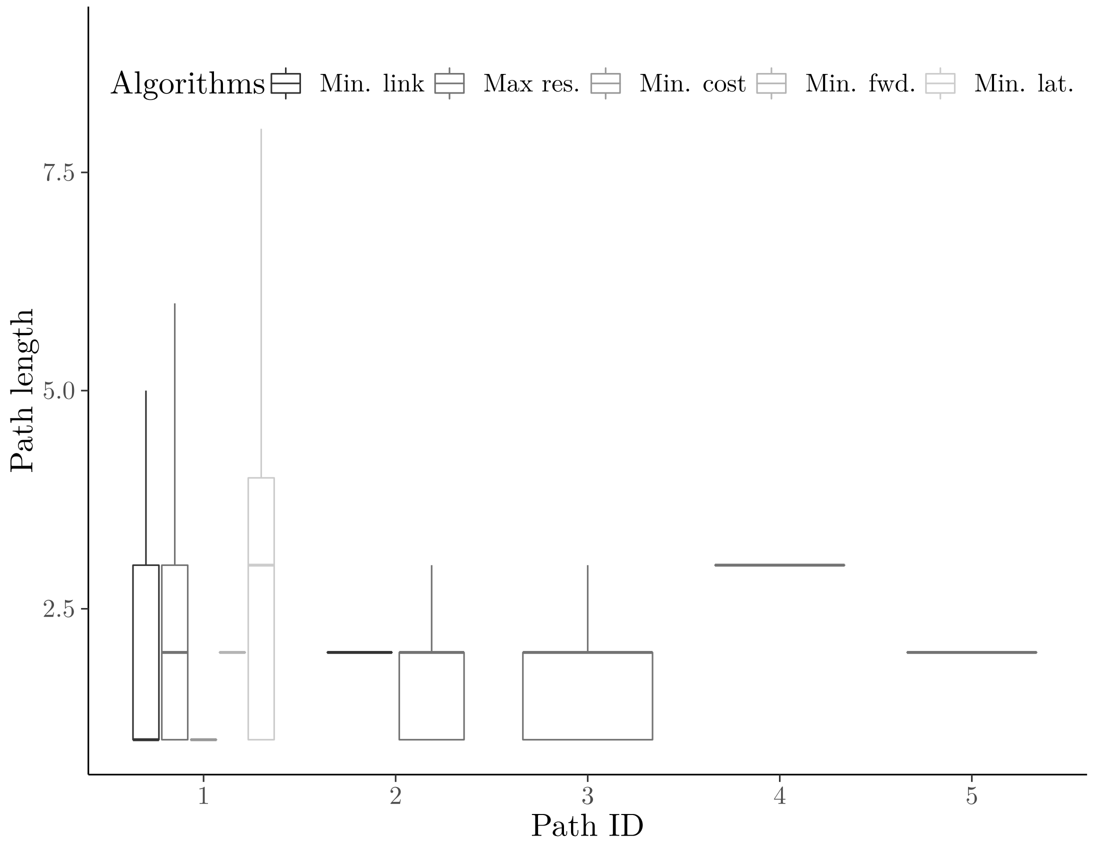


Figure 4.21: Path length of resilient paths using optimisation for the avionics network, outliers hidden for clarity

All other optimisation functions only provide two paths. While some demands can also only be served two demands, the mean case has one additional for the avionics network, while still guaranteeing that requirements with respect to queueing delay and capacity are satisfied. Consequently, in those safety-critical systems where the maximum fault tolerance is desirable, the *maximum resilience* objective function is the most suitable one to calculate configuration templates, even if this results in higher resource utilisation.

4.5 Comparison of Heuristics and Optimisation

The main results presented in this chapter have already been published in [142], albeit not for the network topology presented hereafter. The comparison for resulting queueing delay has already been published in [144].

Table 4.15: Number of Resilient Paths

Network	Nr. of Demands	Min.	Median	Mean	Max.
Avion.	98	3	3	3.351	5
Avion.	196	2	3	3.324	5
Avion.	295	2	3	3.365	5
Avion.	393	2	3	3.349	5
Avion.	492	2	3	3.334	5
Avion.	590	2	3	3.345	5
Avion.	688	2	3	3.367	5
Avion.	787	2	3	3.341	5
Avion.	885	2	3	3.353	5
Avion.	984	2	3	3.339	5
Avion.	1082	2	3	3.34	5
Avion.	1180	2	3	3.354	5
Avion.	1279	2	3	3.356	5
Avion.	1377	2	3	3.351	5
Avion.	1476	2	3	3.348	5
Avion.	1574	2	3	3.357	5
Avion.	1672	2	3	3.357	5
Avion.	1771	2	3	3.35	5
Avion.	1869	2	3	3.352	5

In this section, the results for the best-performing minimisation (*minimum cost*) will be compared to those of the heuristics. This function has been selected as the previous section has shown that good results for the investigated performance characteristics can be thus obtained. Wherever relevant, other optimisation functions will be included in the comparison as well.

4.5.1 Bandwidth

To begin with, the distribution of bandwidth utilisation at the switches is shown in figure 4.22. Here it can be observed that – in most cases – the heuristic requires more than twice the bandwidth at certain switches compared to the optimisation. The only exception to this rule is for $n \leq 295$. Due to the low number of demands and hence low bandwidth requirements, the delay bound of the most critical demands does not yet become a limiting factor. Consequently, the cost minimisation results in cheaper (i.e. less used in the case of the avionics network) paths, an effect that can also be observed in the results for the link utilisation as described below.

As the heuristic uses shortest paths until it is limited by the delay bound of more critical demands, the variance between most and least used switch is also significantly larger.

However, for the avionics topology it also has to be noted that the demands are not equally distributed across the network (cf. table 4.1). Consequently, the variance between utilised bandwidth for those switches that serve primarily as origin/destination and those that are primarily used in transit is larger for this topology than for others. As depicted in figure 4.23, the heuristic utilises more bandwidth at the link than the

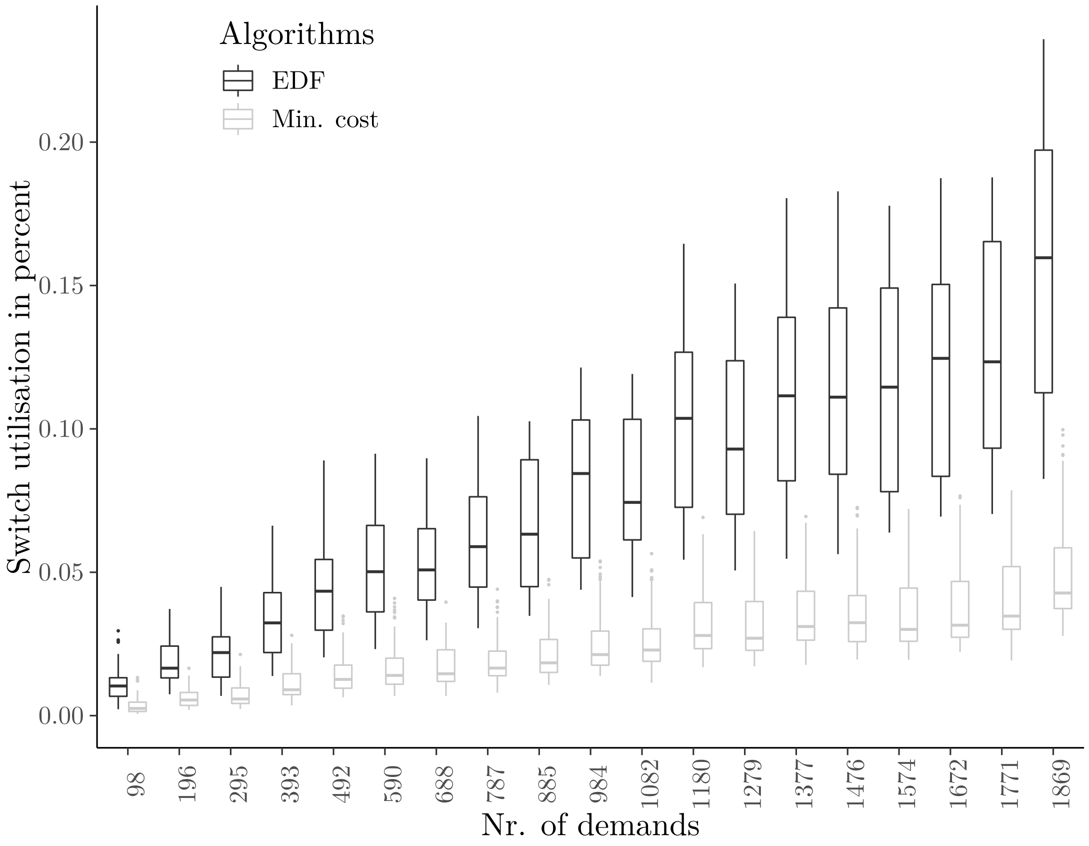


Figure 4.22: Switch utilisation using min. cost and EDF heuristic for the avionics network

optimisation function. As the cost is directly calculated from the link utilisation, the optimisation aims to minimise this parameter. This also results in a smaller increase with the size of the demand set compared to the heuristic. In the latter case, the variance between the least used and the most used link is also greater. As the heuristic algorithm limits the number of additional resources that can be placed together with a timing-critical demand, those paths with less critical demands become more used. Consequently, with respect to utilised bandwidth, optimisation is more suitable to obtain configuration templates as it uses the available resources more efficiently.

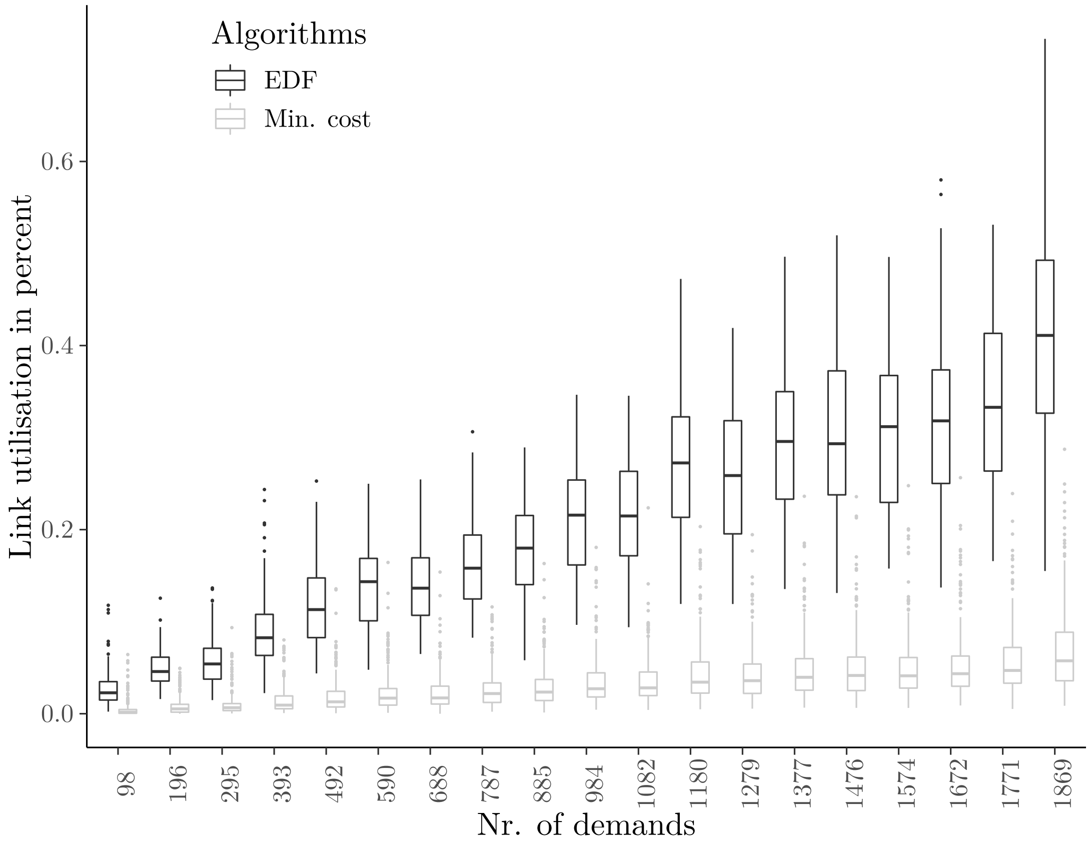


Figure 4.23: Link utilisation using min. cost and EDF heuristic for the avionics network

4.5.2 Calculation Time

The comparison of calculation time is given in figure 4.24. For clarity, this plot has been cropped, consequently one outlier for $n = 196$ demands is not depicted. Overall, it can be observed that the optimisation takes significantly longer to be calculated. Especially, with an increasing demand set, the problem becomes harder to solve optimally, thus increasing the calculation time. While the calculation of the heuristic also increases with the size of the demand set, the maximum calculation time is slightly more than half a second, so it would not preclude online deployment. The optimisation, on the other hand, requires several minutes. In addition, the presence of an outlier with 2792.04 seconds at $n = 196$ shows that constraints of individual problems have an even more significant impact on the calculation time. As an average calculation time is thus hard to predict, an online deployment is unrealistic in safety-critical systems.

Hence, heuristics are more suitable to obtain configuration templates quickly.

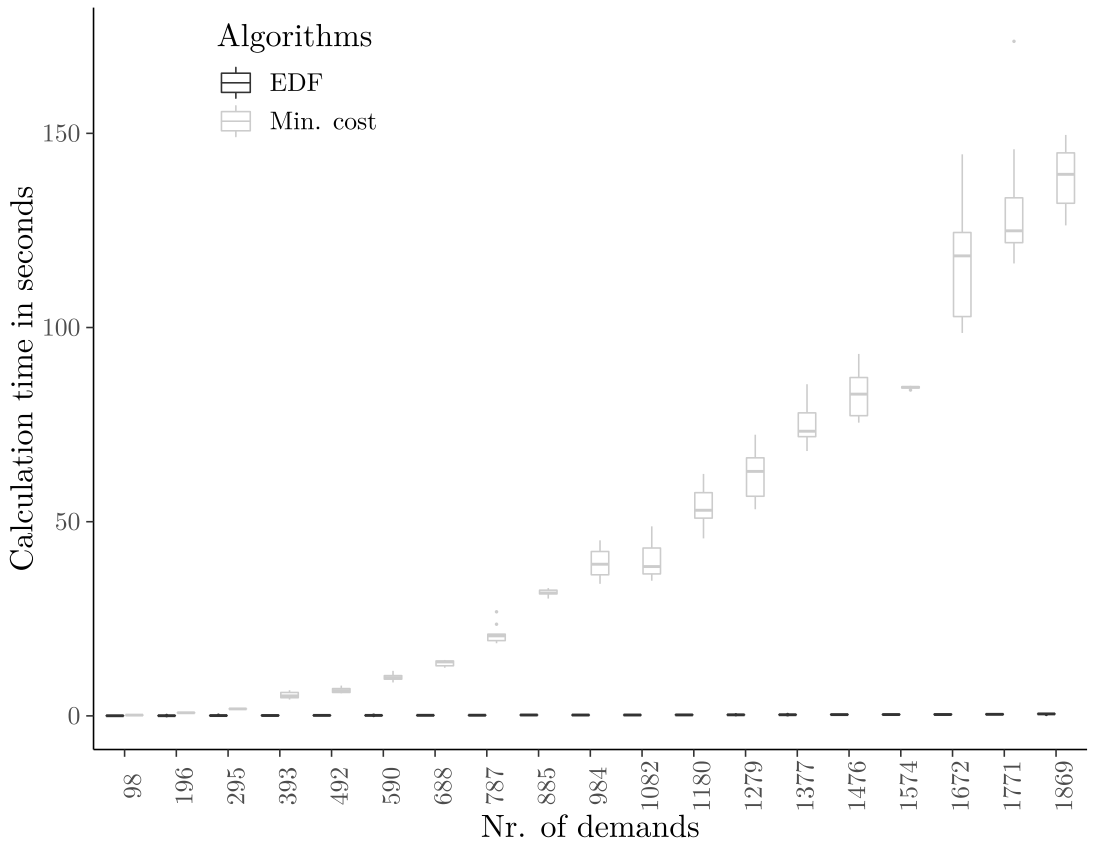


Figure 4.24: Calculation time using min. cost and EDF heuristic for the avionics network

4.5.3 Cost

The distribution of cost for the avionics network is depicted in figure 4.25. It can be observed that the heuristic results in higher cost for all sizes of demand set. What is more, for the optimisation there is only a factor 6.63 between minimum and maximum cost (i.e. between smallest and largest demand set). In contrast, for the heuristic, the factor is 523.04, with a larger distribution within the demand set. Here, it can be observed that less critical demands need to accept longer paths to prevent negatively affecting more critical paths on the shortest path. As the cost is proportional to the path length for this network, the cost optimisation also produces shorter paths than the heuristic.

Thus, with respect to cost, optimisation yields cheaper configuration templates.

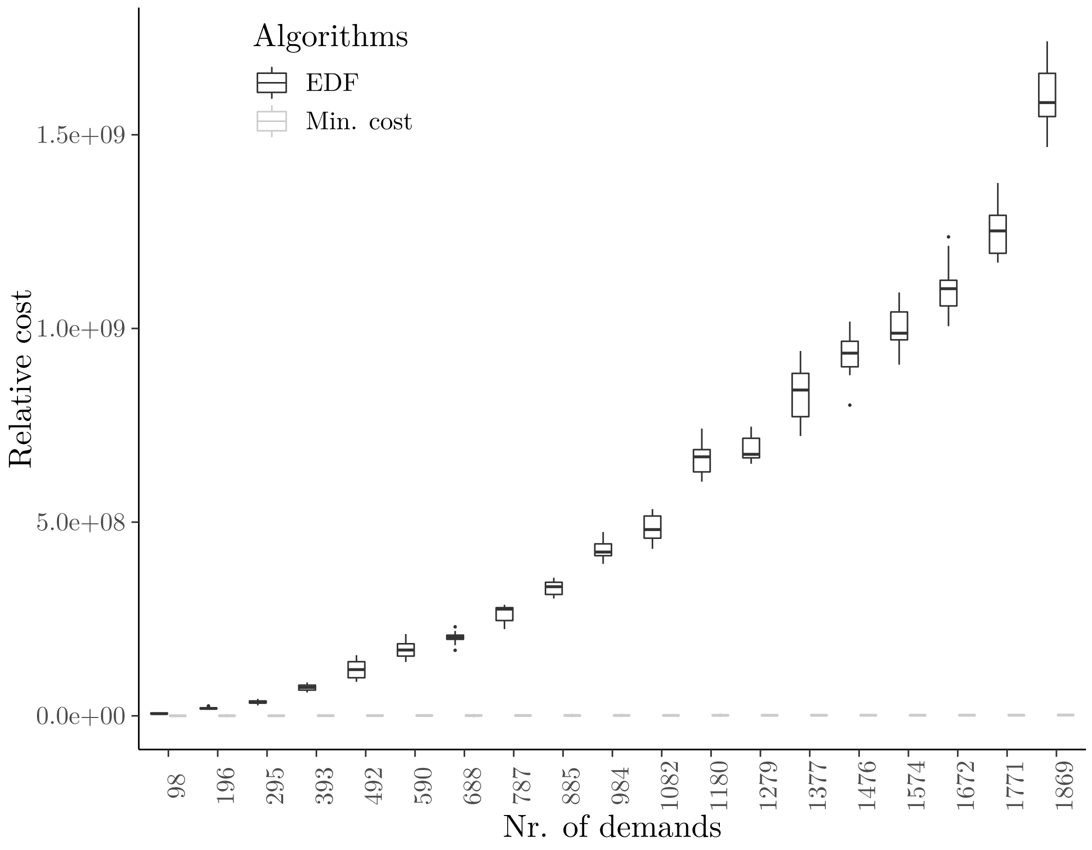


Figure 4.25: Cost using min. cost and EDF heuristic for the avionics network

4.5.4 Delay

In figure 4.26, the variation of the queueing delay with the size of the demand set is depicted. In addition, figure 4.27 gives the factorisation for primary and secondary paths. For $n \leq 295$, the optimisation gives a higher maximum latency than the heuristic. In those cases, only very few demands are critically bound and the resource usage is not high enough to trigger the delay bound, as there are also no violations for the heuristics in those sets. Consequently, the cost minimisation objective becomes dominant thus yielding longer paths.

Yet as more demands need to be routed, the delay bound limits the opportunities of the optimisation algorithm to limit cost and in turn also the latency. Hence, the increase in the size of the demand set for the optimisation algorithm are far less significant for the optimisation than for the heuristic.

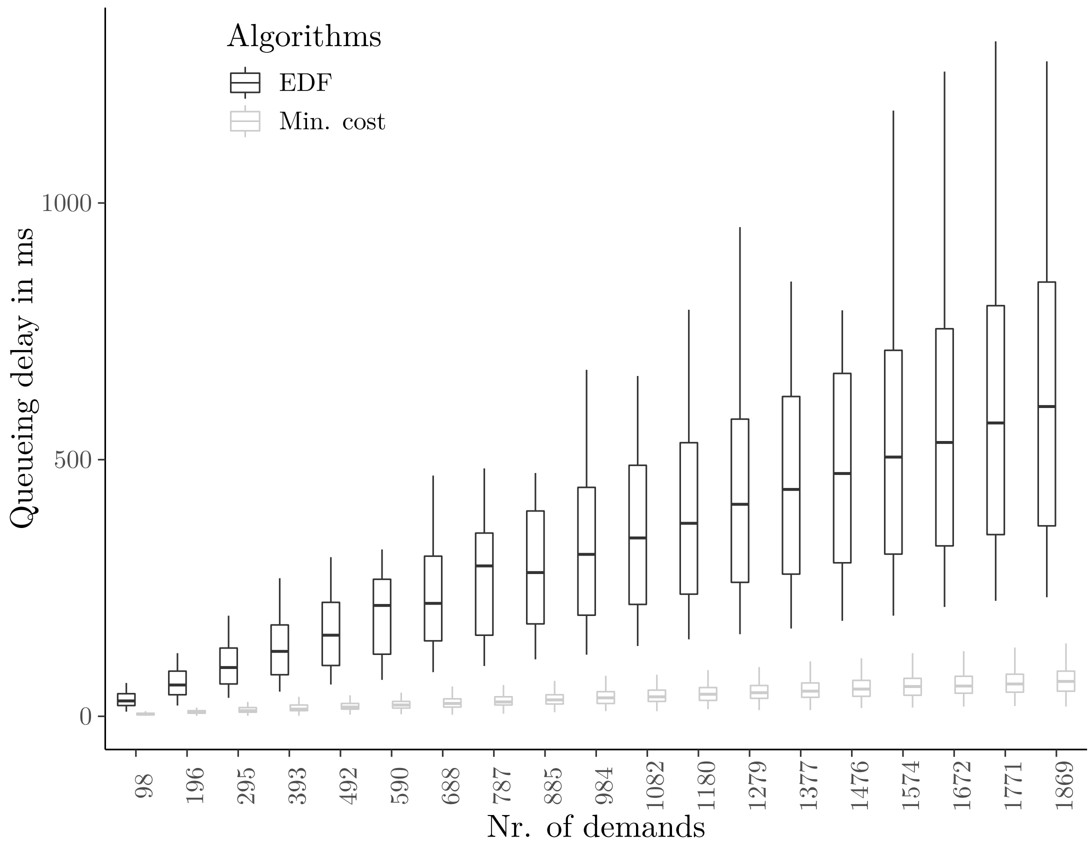


Figure 4.26: Queueing delay using min. cost and EDF heuristic for the avionics network

Figure 4.28 shows the density function for the avionics network in the comparison between *min. cost* and the EDF heuristic. This figure illustrates the much higher density of very low delays using the optimisation function compared to the heuristic. Still, the optimisation function allows to consider the various constraints and prevent violations, unlike the heuristic.

4.5.5 Forwarding Table Entries

With respect to the distribution of forwarding rules on the switches, the difference between heuristics and optimisation is even more pronounced than for the bandwidth optimisation, as shown in figure 4.29. The effective lack of a delay bound for $n \leq 295$ again affects the optimisation by yielding slightly higher values for the smallest demand sets. Yet for the demand sets beyond, the optimisation very slightly increases the number of forwarding rules required.

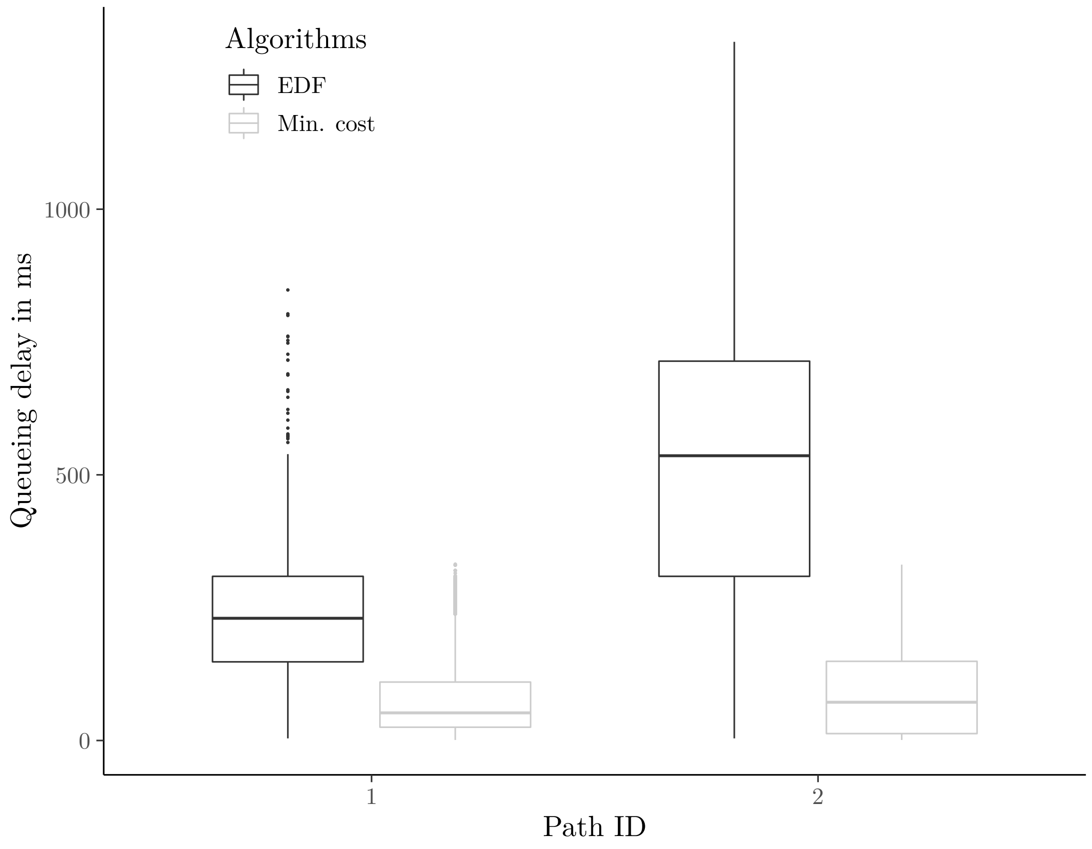


Figure 4.27: Queueing delay using min. cost and EDF heuristic for the avionics network for $|D| = 1869$

In contrast, the number of forwarding rules increases exponentially for the heuristic. While not a problem for the demand sets present, for larger demand sets it could quickly become a problem, as the TCAM memory is limited to about 8000 rules. For the largest demand set in the avionics network, the heuristic required up to 4.5 times the number of forwarding rules than the optimisation. Consequently, the optimisation is better suited to obtain configuration templates with respect to efficient use of TCAM.

4.5.6 Path Length

As depicted in figure 4.30, the mean path length for the heuristic is twice that of the optimisation, even if the maximum path length is greater for the latter. As the size of the demand set does not have a noticeable impact on the path length, the corresponding plot has not been included for clarity. However, with increasing demand set size, the impact of the longer path becomes more significant on the other parameters investigated, e.g. cost or queueing delay. In contrast, while a few (less critical) demands have to accept longer paths with the optimisation, the effect on overall algorithmic performance is

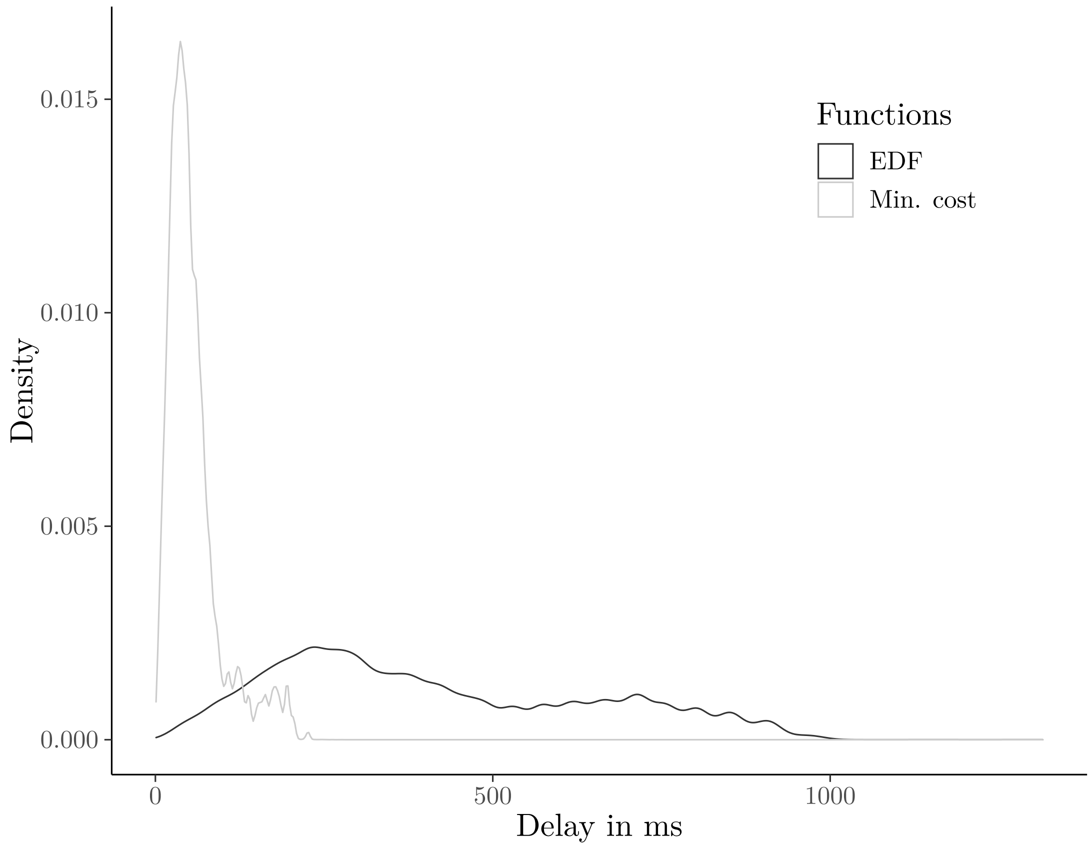


Figure 4.28: Latency distribution using min. cost and EDF heuristic for the avionics network for $|D| = 1869$

negligible. Consequently, using optimisation is more suitable to generate configuration templates with short paths.

4.5.7 Violations

As depicted in figure 4.31, the inclusion of latency constraints is also relevant for optimisation functions. Here, the *min. link utilisation* function has been performed without latency constraints. While no violations were recorded for either algorithm for the two smallest demand sets, the EDF algorithm already violates constraints at $n = 295$ demands. As more demands need to be placed, the number of violations also increases. Interestingly, the optimisation function gives – on average – fewer mean violations until $n = 1476$ demands. As the cost for the optimisation is defined as a step function, an increased demand set will result in more demands being placed on a link until the next step in the cost function is reached.

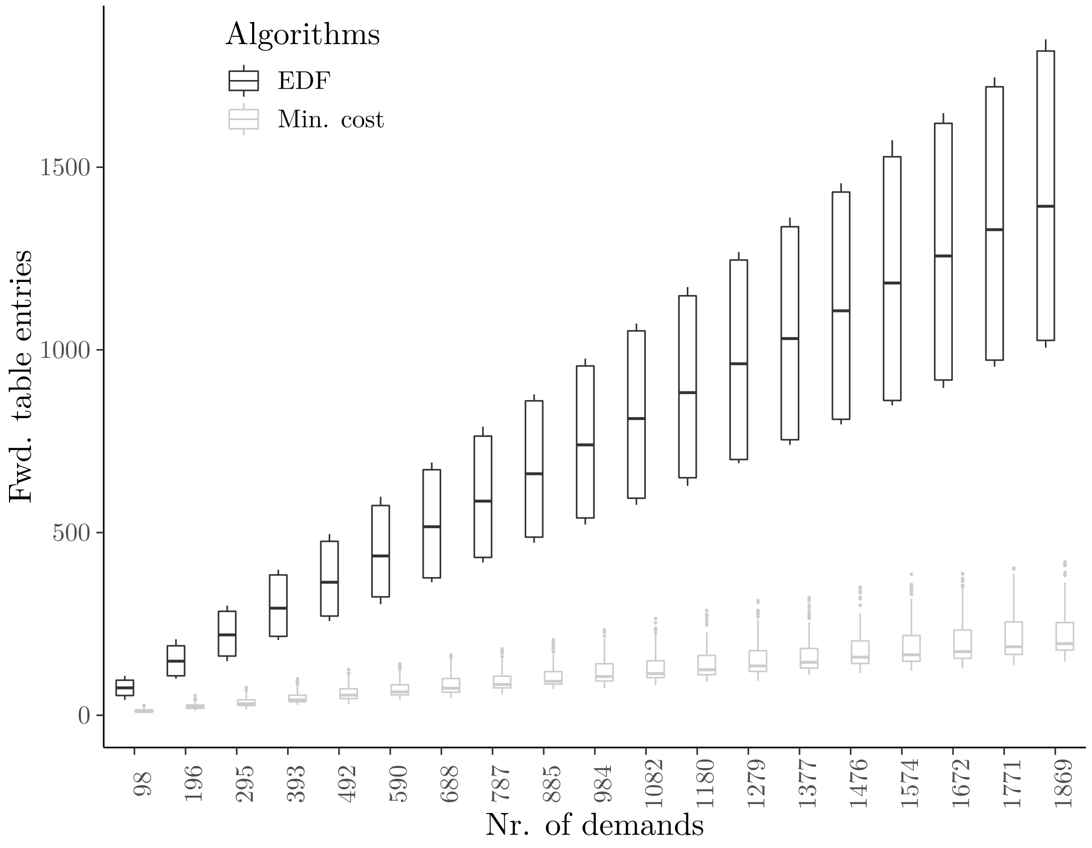


Figure 4.29: Forwarding rules using min. cost and EDF heuristic for the avionics network

Consequently, the number of critical demands that are being violated increases likewise. Yet as the demand set is not equally distributed across the network, the violations by Dijkstra's algorithm eventually level off. On the other hand, the *min. link utilisation* aims to equally distribute additional demands equally across the network (to keep as many links as possible at the lowest cost). Consequently, the maximum number of violations is nearly 60% higher for the optimisation than for the EDF heuristic when the optimisation does not consider the maximum delay.

Hence, when real-time traffic needs to be transported in a safety-critical system, it is absolutely necessary that the according constraint is included into the optimisation problem. While this significantly increases calculation time, it is the only way to ensure that templates can satisfy the timing requirement of safety-critical traffic.

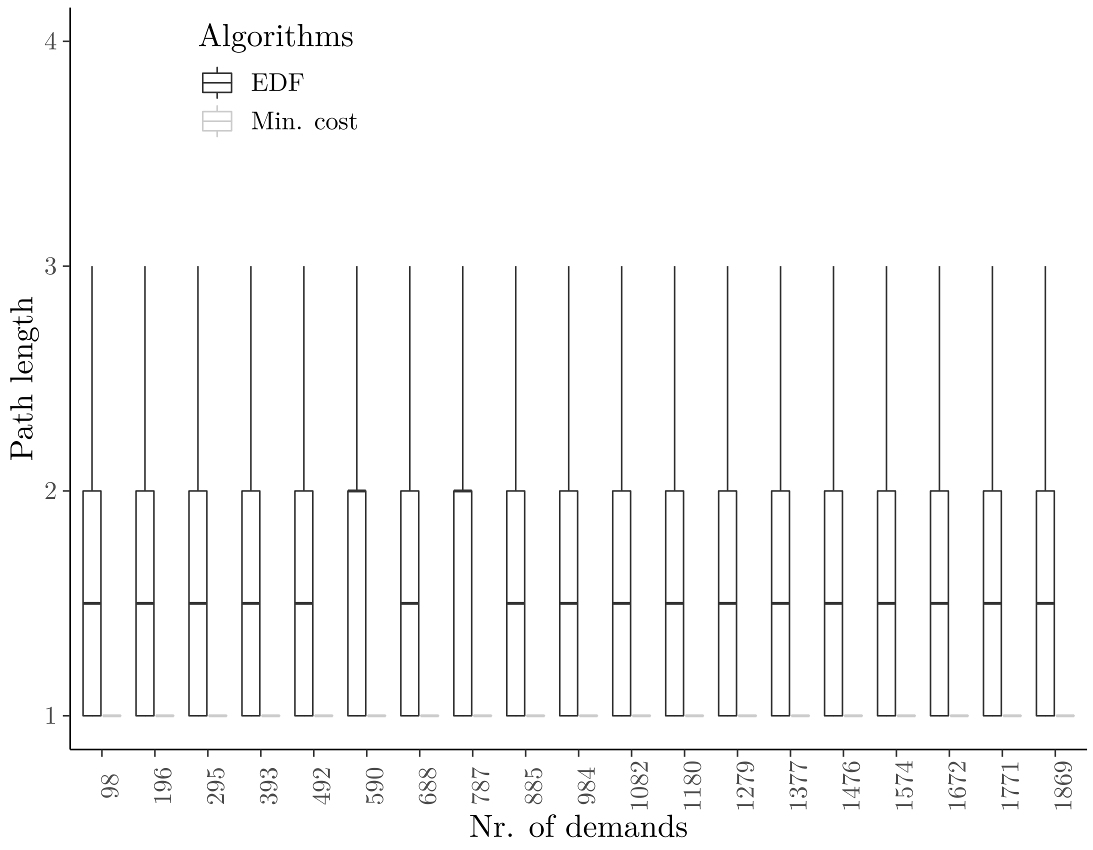


Figure 4.30: Path length distribution using min. cost and EDF heuristic for the avionics network

4.5.8 Effect of Varying Delay Bounds

As mentioned above, for some network topologies, the effects of varying the delay bound have been studied. However, if at least one problem could not be solved without violating some constraints, it is not included in this analysis.

Optimisation

Calculation Time: One of the most significant effects could be observed for the calculation time. As an increasing number of time-critical flows need to be routed, the problem becomes more complex to solve, thus increasing the calculation time. This effect could also be observed for the avionics network with the number of demands, even though no specific variation of latency constraints has been performed.

Resilience: Interestingly, no discernible impact on the number of available resilient paths has been observed.

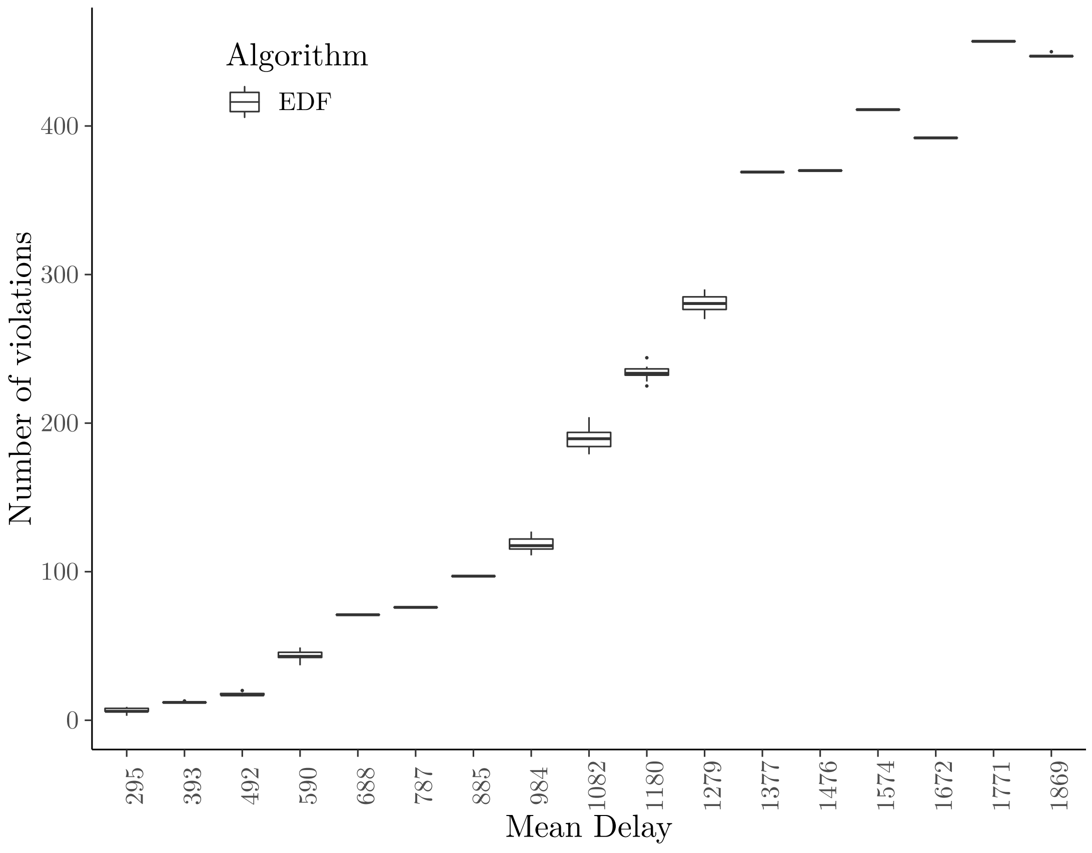


Figure 4.31: Violations using min. cost and EDF heuristic in the avionics network

Queueing Delay: For the DFN network, an impact on the queueing delay could be observed for the *maximum resilience objective function.* As the tighter delay bounds reduced the number of resilient paths, the queueing delay reduces in turn.

In addition, for the ATLANTA network and the *min. latency* objective function, a decreasing mean delay bound also resulted in a decrease of the queueing delay. However, other topologies – while displaying some variance – did not exhibit such behaviour. Consequently, with respect to the queueing delay, the impact of a specific topology is at least as significant as that of specific delay bounds.

Heuristics

As neither Dijkstra nor the shortest path heuristic consider the delay bound, it is not expected that the varying latency constraints have any impact on performance. The only conceivable impact would be on the number of violations. This has been observed for the ATLANTA network as depicted in figure 4.32. Moreover, until a mean delay bound of 840 ms, only the link capacity constraint is being violated, while the tightest constraint investigated $\mu = 630$ also experienced a significant number of delay bound violations. Similarly, for the Nobel-Germany network, the only violations occur for the tightest mean delay bound $\mu = 484$ (delay bound violations). However, as shown by the variation between the different experiments for the EDF functions, individual problems have a more significant impact on the number of violations. Notwithstanding, especially for the EDF functions, tighter constraints are significant, as eventually no further demands can be placed without violating constraints, while the optimisation functions can consider all constraints.

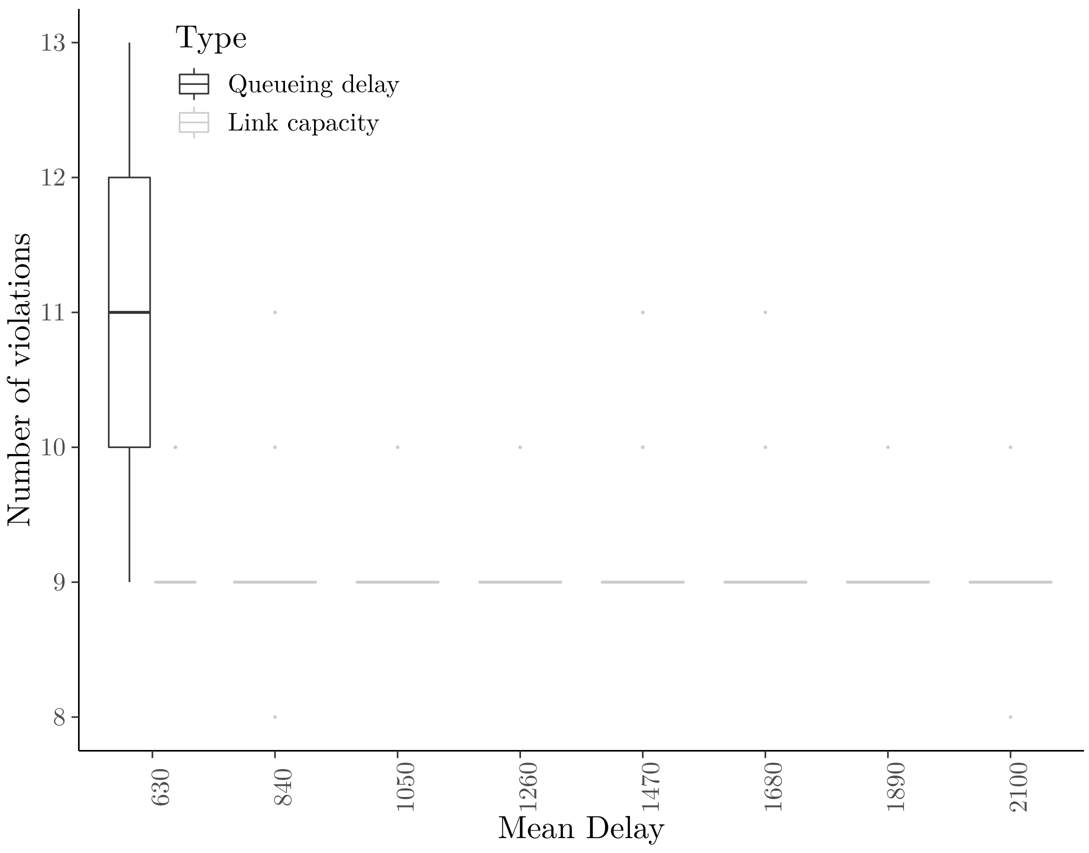


Figure 4.32: Distribution of constraint violations for the ATLANTA network

To summarise, while heuristics provide network configurations quickly, they result in violation of timing or capacity constraints and are thus unsuitable for critical systems.

However, due to the significant computational resources required in solving optimisation problems, they cannot be performed upon detection of a failure or security incident.

Hence, candidate routings together with other possible reactions to anomalies need to be established before a system becomes operation.

Yet such candidate routings can and need to be changed during runtime of a critical system, which may well span several decades. Consequently, a mechanism needs to be developed to ensure that safety-critical systems have suitable candidate responses at their disposal when anomalies are detected.

4.6 Generating Templates

For most of today's IT systems, it does not matter if transmission is briefly interrupted or some of the traffic is lost. Contrary to that, if networks of safety-critical systems were to drop some traffic during reconfiguration, it could have dire consequences, e.g. if a braking signal or a shutdown command for a power plant is lost or delayed.

Consequently, due care is needed during reconfiguration to prevent the loss of traffic for those parts of the network that are unaffected by the failure.

As shown in the previous chapter, heuristics cannot be applied to generate network configurations. Rather, the routing problem needs to be formulated as an optimisation problem. While the *minimum cost* function yielded very good results with respect to network performance, it only provided one backup path. To ensure that a failure mid-transmission does not cause traffic to be lost, both the standard and the backup path need to be served concurrently e.g. through flow replication. Thus, while one failure would be tolerable through such means, more complex failure scenarios as they occur during cyber attacks could not be addressed.

Consequently, the results provided by the *maximum resilience* objective function are used as it provides the maximum number of disjoint paths and thus the highest number of faults that can be mitigated.

While the basic concept of configuration templates has already been published in [144], this thesis provides a more detailed description. Specifically, it advocates the need for such templates in a more concise way and provides the distinction from fail-safe states. In addition, it explores some further usages of heuristics, and thus goes above and beyond the discussions in the paper.

4.6.1 Running vs. Possible Configurations

However, it is important to make the distinction between a *running* configuration and the set of disjoint *possible* configurations that are thus obtained. This is due to the safety-critical nature of the applications for which they have been designed.

As detailed in previous chapters, the *maximum resilience* algorithm provides the maximum number of resilient paths for each demand. Consequently, it may be the case that not all flows have the same number of resilient paths available. Furthermore, while creating individual templates for single failures is possible, it would be increasingly hard to create templates for successive failures. This would require a plethora of different configurations that consider all possible combinations, and would consequently also require significant memory. To illustrate this, table 4.16 shows the number of network configurations required to address every possible failure case for up to five failures in the avionics networks.

Table 4.16: Required Configurations for Successive Failures in the AFDX Network

Failures	**Templates required**
1	24
2	552
3	12,144
4	255,024
5	5,100,480

In addition, as no traffic must be lost during reconfiguration, a possible configuration may not be applied directly unto the network. Rather, due care needs to be exercised in order to ensure that the transfer from one running configuration to the other is smooth.

4.6.2 Configuration Templates

Consequently, the concept of *configuration templates* has been introduced. A template is the collection of possible paths that are guaranteed to not interfere with each other and satisfy all requirements. Here, the results of the optimisation problem are *templates* of configurations, and not configurations that may directly be applied on the network. Depending on previous failures and on the current network state, forwarding rules are deleted on some nodes while new ones are installed.

Consequently, the paths provided by the *maximum resilience* objective functions are divided into two distinct *configuration templates*:

Standard The default path, e.g. the shortest path, lowest latency etc. and n backup paths used to ensure that the required failure tolerance is reached.

Remainder Any remaining paths are put in this template to be available in case of failures.

Here, the paths that are included in the **standard** template are served concurrently from system initialisation onwards. Thus, the safety-critical traffic continues to flow over the unaffected route if a failure occurs. This allows sufficient time to select a suitable path of the **remainder** to restore the resilient path. Consequently, the safety-critical traffic continues to be transported uninterrupted and reconfiguring the backup path allows that there are again two resilient paths available.

This process can be repeated for successive failures until no more paths are available in this template. Only failures after this point result in performance degradation, because sufficient resilience against *future* failures is ensured.

Formally, the maximum resilience algorithm provides $P_{r,d}$ paths where $|P| = k$. If a failure f is detected, one such path is selected so that $\forall f \in F : f \notin P$.

4.6.3 Configuration Templates Versus Fail-Safe States

Technically, to a limited extent a similar approach is already taken for most critical systems. *Fail-safe* states are presently used to safely transition into a system state where critical functions continue to work, albeit in a possibly degraded mode. This is in line with a requirement derived from availability, *graceful degradation* [160]. Thus the system does not fail suddenly and completely but rather gradually reduces functionality in order to preserve as much of the system's critical functionality as possible.

Fail-safe states address specific failure cases or attacks. For example, alternate control laws can be introduced that allow more control input from operators if computers cannot automatically cope with multiple sensor failures. Another option is to simply drop everything except the most critical traffic at switches.

Current certification regulation e.g. [48] requires that all possible failure conditions should be thought of and mitigated before the system enters into service. However, this approach is not feasible for cyber threats as new attack vectors may emerge during system operation. Consequently, a more dynamic approach is needed.

4.6.4 Templates and Heuristics

However, as mentioned in the introduction, not all traffic in a safety-critical system is also safety-critical. Even if critical networks are segregated from non-critical networks, information related to diagnostics and monitoring may still be required to be transported. While it is possible to arbitrarily define very high delay bounds (in the second range), they might still have a detrimental effect on safety-critical traffic.

Consequently, one option would be to only consider safety-critical traffic for the calculation of configuration templates. Previous sections (section 4.3) have shown that heuristics are unsuitable to calculate configuration templates. Notwithstanding, after a failure, heuristics such as EDF could be used to route non-critical traffic while ensuring non-interference with safety-critical traffic. In order to do this, the heuristics presented in this thesis would have to be adapted so that only non-violating paths are posted. Consequently, not the complete set of non-critical demands would be routed, but only those where non-interference can be guaranteed.

This would serve as a complement to existing strategies such as traffic prioritisation through VLAN tagging. As it has been shown that significant spare resources exist, this would enable to use them more efficiently, even in the presence of failures.

Furthermore, the heuristics need to be carefully designed with respect to functionality, especially if several aspects are being combined (e.g. delay bounds with capacity limits). Previous chapters have shown that considering only one or the other may lead to violations of capacity or delay bounds.

Consequently, in some critical systems (especially those that need to be certified), even non-determinism of uncritical traffic may not be acceptable due to the potential impact on critical functions.

4.6.5 Certification of Configuration Templates

The aim of the *configuration templates* presented in this chapter is to present generic reactions to detected anomalies that aim to mitigate failure cases that have not been considered specifically during system design. Hence, they work as an extension to fail-safe states, but without necessarily reducing system performance. When a fault or attack has been detected, a different template not using the affected component can be applied on the network, thus allowing a continued safe operation.

Consequently, the process of generating configuration templates is as follows:

1. Maximum resilience optimisation problem is solved for a given network configuration
2. Necessary certification authorities certify that templates are safe
3. Templates are loaded into a template storage and checked for access by critical system
4. (Templates are withdrawn from the template store if they are outdated)

To begin with, templates are generating by solving the *maximum resilience* optimisation function as described in the previous chapter. Depending on the system, not all constraints may be relevant, e.g. the latency constraint may not be relevant for a smart grid because it operates on different timescales than a real-time vehicular network.

Secondly, the certification authorities investigate the safety properties of the network configurations. This process is comparable to current certification methodology. However, rather than using paper-based accountability processes, digital processes such as DLT are being used.

Thirdly, the templates are loaded into a database that is accessible by the SDN controller. Obviously, prior to loading a variety of parameters are being checked, such as the validity of signatures and the applicability to the system.

Additionally, configuration templates may also be outdated, e.g. because components are added or removed during the decades of operation of most safety-critical systems. Hence, a mechanism to remove such outdated templates also needs to be included. An overview of this process is provided in figure 4.33. This ensures that the system

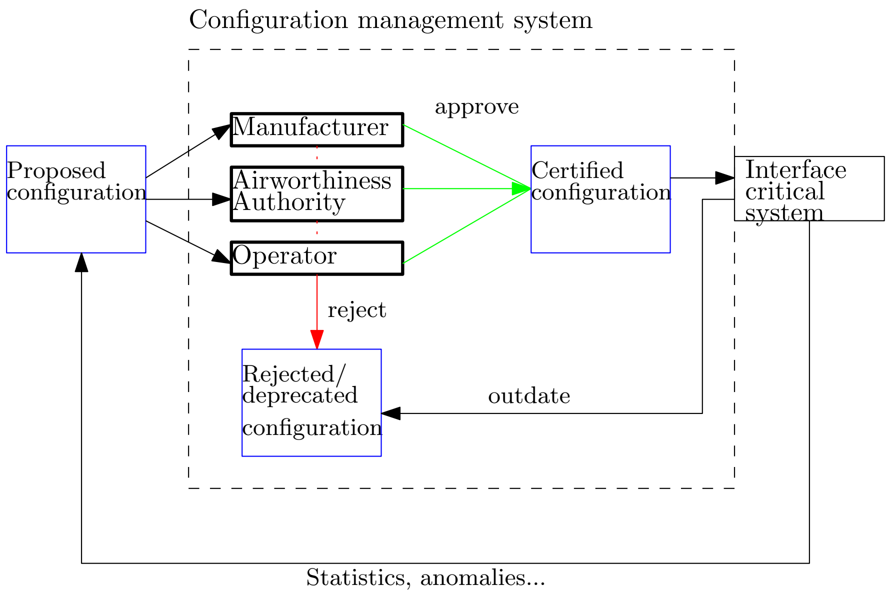


Figure 4.33: Certification process

always has configuration templates available that can provide maximum resilience to the network. In addition, a similar process could be envisaged specifically to react to security incidents, which will likely require more frequent updates.

4.7 Summary

To summarise, while heuristics can result in good performance with respect to some characteristics and also provide fast routing, their violation of timing constraints make them unsuitable for use for safety-critical traffic demands. Beyond that, it has been shown that the EDF heuristic results in violations of the resource usage while the *lim.*

capacity heuristic causes delay bound violations. Consequently, for safety-critical traffic, neither heuristic is suitable.

However, this chapter has also shown that due to the significant resources required in solving optimisation problems, it is not feasible to perform them only when a failure or security incident has been detected. Consequently, candidate routings together with other possible reactions to anomalies need to be calculated before a safety-critical system comes in operation. While a variation of the delay bound has been investigated, it was found that the most significant impact is on the calculation time. For the other parameters investigated, particularities of a given routing problem have a more significant impact on performance than the mean delay bound.

In addition, this chapter has also shown that is infeasible to create a tailored configuration for every possible failure. Consequently, the use of *configuration templates* has been presented. To this end, an optimisation algorithm for *maximum resilience* has been employed to generate candidate responses that can then be selected as needed during operation. Finally, a possible application of heuristics for best-effort placement of non-critical demands has been presented.

Chapter 5

Using Configuration Templates

This chapter describes how the configuration templates can be used to react to detected anomalies. To this end, the templates are used to react to intentionally triggered failures and the effect of this process studied, as detailed in section 5.1. Subsequently, the impact of using configuration templates on the management plane are given in section 5.2, with safety-related considerations in section 5.3 and security-related in section 5.4. The main findings of this chapter are summarised in section 5.5.

5.1 Network Reconfiguration

The experimental setup in which the data plane reconfigurations were performed are given in section 5.1.1.

To analyse the impact of network reconfigurations on the control plane, different properties will be investigated. To begin with, the impact of the failure (i.e. the number of flows affected by a failure) is studied in section 5.1.2. Whether the anomaly could be mitigated successfully, and how long it takes until all demands are rerouted is investigated in section 5.1.3 and section 5.1.4, respectively. Additionally, the potential impact of the number of previous failures on those parameters will be investigated in section 5.1.5.

Furthermore, the amount of control traffic between controllers and switches will be analysed in section 5.1.6 in order to determine whether in-band or out-of band control networks make more sense in this context.

Beyond that, strategies and the impact of deleting forwarding rules prior to the reconfiguration are discussed in section 5.1.8. A condensed version of this section has been published previously in [145]. However, the paper only provides a basic overview of the concept, but does not include the detailed analyses that are presented in this thesis. Specifically, the paper does not address the impact on strategies to delete rules and control traffic necessary for the proposed solution.

5.1.1 Experimental Setup

The avionics network has been replicated in the OpenSource `mininet` network emulation orchestration system [123]. End-hosts, links, routers and switches are virtually executed on a single Linux kernel. It has been run natively on a machine running Ubuntu 16.04, with 48 GB of RAM and 12 cores due to the large number of traffic flows.

Mininet is started as a python shell script (cf. Listing B.1). It initialises the network, and creates four hosts per demand. Two of those represent the source of the demands attached to the source switch in the network and the other two are attached to the last switch in the network. Subsequently, the appropriate forwarding rules from the initial configuration as described in the previous chapter are installed on the respective switches.

As soon as this has finished, switches and/or links start to arbitrarily fail. To this end, a small attacker program has been implemented in `python` that reads the network topology and then terminates a random element (cf. Listing B.3). This process is encased in an infinite loop, with a random waiting time between events.

A third `python` script acts as an adaptive controller (cf. Listing B.2) that is built on top of a standard POX controller. This script reacts to the OpenFlow events detected by the controller. Firstly, the script detects which flows (both of the standard and the backup path) have been affected by the failure. To this end, it iterates over the initial configuration template.

Secondly, the candidate paths are evaluated. Here, it is checked whether any previous failures could render a candidate response unusable or whether the failed component is a source or destination switch. If this is the case or no candidate paths without failed components can be obtained, the demand is unrecoverable. Otherwise, the forwarding rules leading to or from the failed component are deleted (see section 5.1.8) and the forwarding rules necessary for recovery are installed.

For installation, the controller orders the modification of forwarding rules at the respective switches with an individual update for each forwarding rule. As the paths are required to be node-disjoint, strategies for rule-sharing are not beneficial in this context. Here, the rules on the switches that are directly connected to the respective hosts (a_d, b_d) are modified first. Subsequently, the rules for intermediate switches are updated. The respective switch reports the successful installation of the new forwarding rule. Subsequently, now redundant forwarding rules are deleted from the switches. While this latest step is important to keep forwarding tables as short as possible, it is not itself time-critical, as there is again redundancy. Consequently, the duration of this final step has not been included in the timing analyses below.

5.1.2 Impact on Demands

Figure 5.1 shows the distribution of demands affected by a failure. For the purpose of this analysis, failures of links have been considered unidirectionally, that means a failure of the link (2,4) will not affect the traffic on link (4,2). It can be observed that a link failure affects far fewer demands than a switch failure.

Contrary to the scenario chosen for deriving the templates which focused on avoiding a failure of the physical link, during operation, failures due to overuse or DOS are far more likely. As those do not influence traffic flow in both directions, it is three times more likely that a failure affects a given switch than a given link. Consequently, the mean number of flows disrupted by a link failure is approx. 10.26%, while this value rises to 62.05% of demands that are impacted. Please note that the numbers may exceed 100% for the switch failure, if a source/destination switch is affected.

Yet as there are end-devices attached to the switches, a failure obviously affects all demands originating from or destined for those switches.

5.1.3 Success of Configuration Changes

Beyond that, figure 5.2 shows the number of new forwarding rules that need to be installed to recover the standard path, while the distribution for the backup path are shown in figure 5.3. As depicted, the restoration of the backup path requires – on average – about four times as many new forwarding rules. This is due to several reasons. Firstly, switch failures can only be recovered if neither the source or destination switch has failed. As the standard path is mostly the direct connection between source and destination, it cannot be affected by a failure of a transit switch, which is not the case for the backup path.

Secondly, due to the requirement that paths are node- and link-disjoint, a completely new path is selected, rather than just a local detour around the failed switch. This in turn leads to more forwarding rules that need to be updated in order to restore the backup path compared to the standard path.   An analysis on the number of flows that could not be recovered is given in figure 5.4 and figure 5.5. As shown in table 4.1, no switch in the avionics network is only used for forwarding. Consequently, all traffic

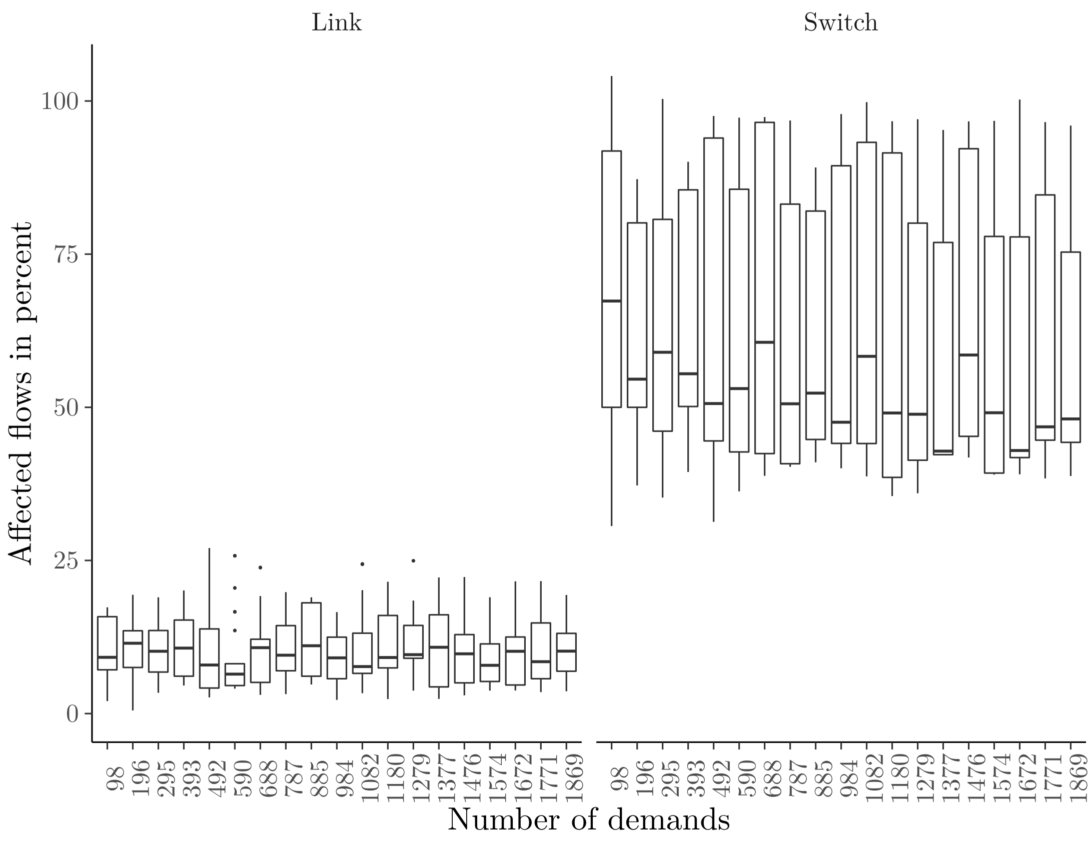


Figure 5.1: Percentage of flows affected by a failure in the avionics network

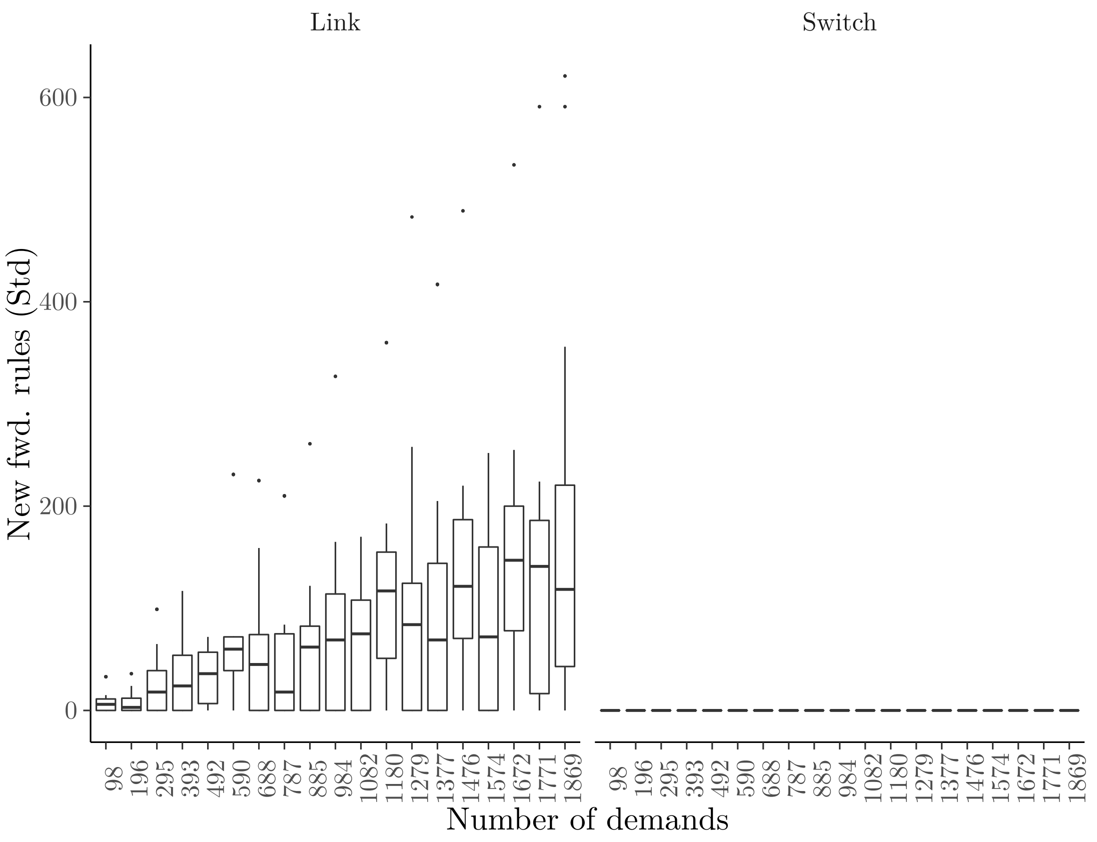


Figure 5.2: Number of new forwarding rules standard path in the avionics network

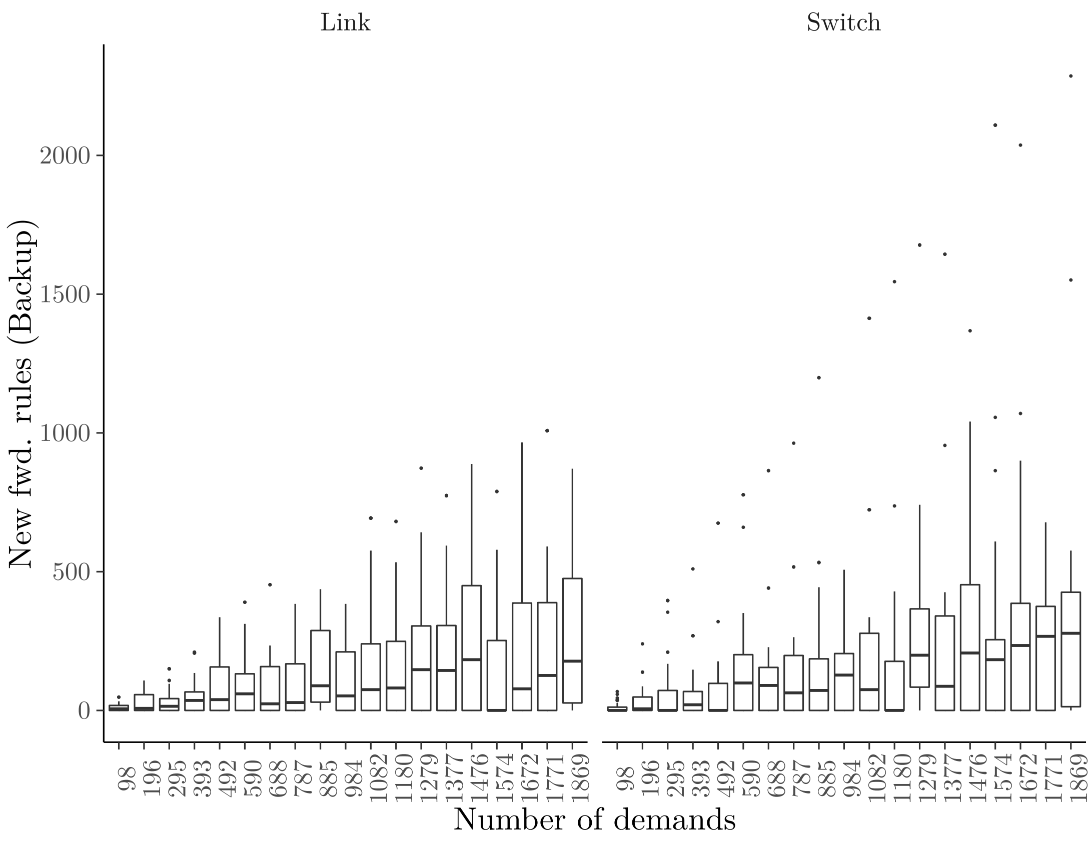


Figure 5.3: Number of new forwarding rules backup path in the avionics network

that originates from or is destined for a switch that is affected by a failure cannot be restored. In addition, as not all demands have the same number of paths available, a previous failure might have already severed the original link. The effect of such previous failures is discussed in section 5.1.5.

Notwithstanding, it should be noted that yet another failure on the final running path would finally sever the link between source and destination.

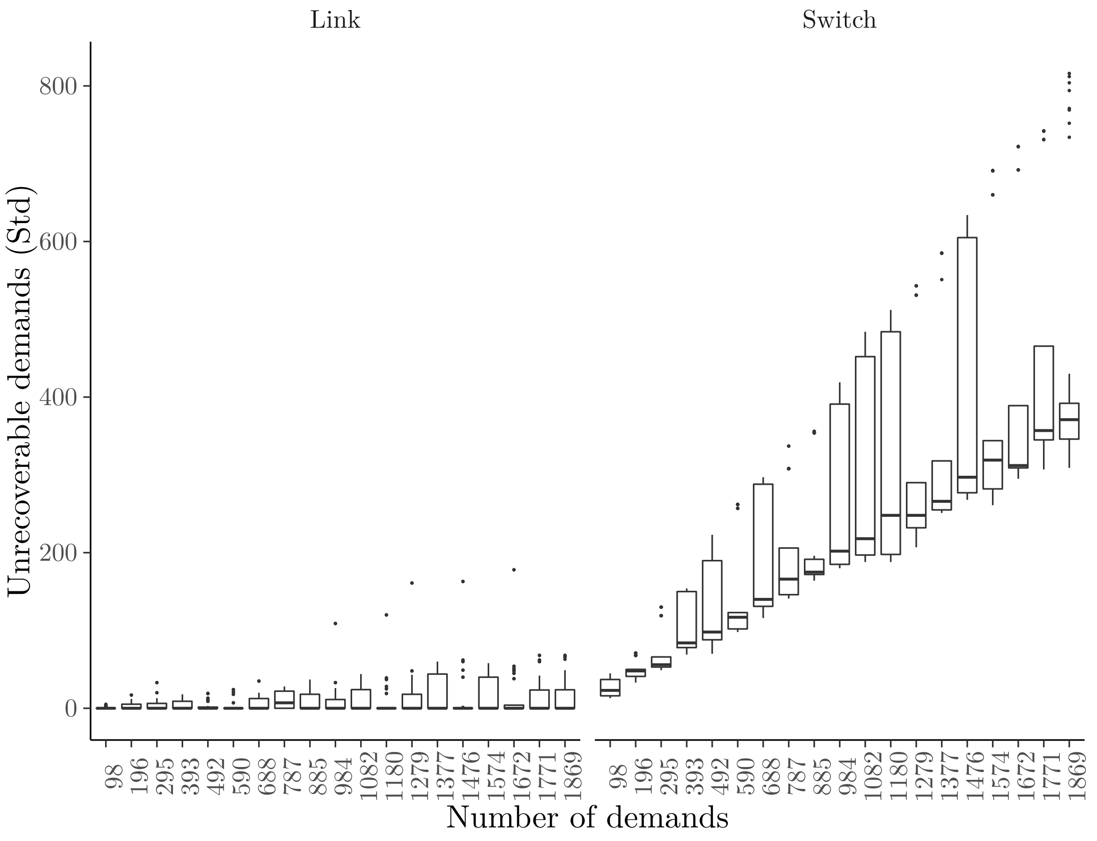


Figure 5.4: Number of unrecoverable flows in the avionics network – Standard configuration

5.1.4 Time Required

In figure 5.6 the time until successful reconfiguration is depicted. It should be noted that this is the time required until all flows affected by a failure have been rerouted, not the time for individual rule installation. In addition, the time measured here is the elapsed time until the instructions have been sent to the switches by the controller, not until they have reported the successful installation back to the controller.

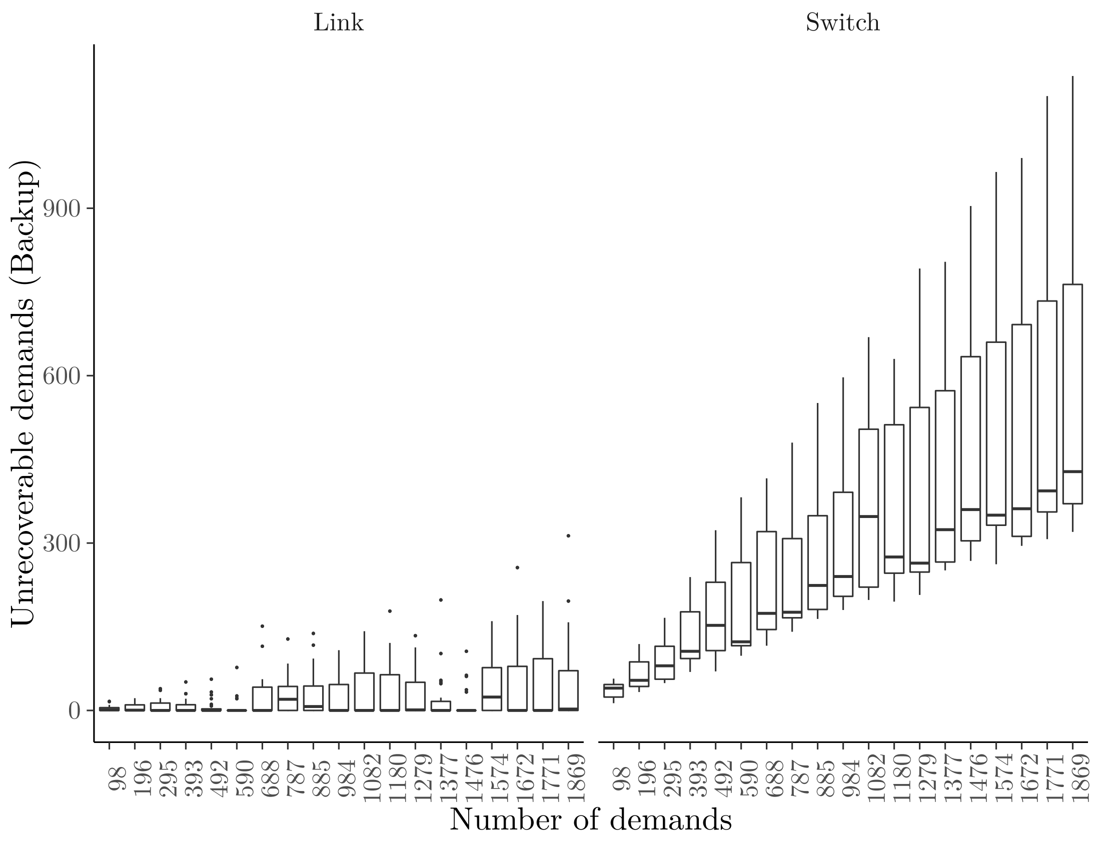


Figure 5.5: Number of unrecoverable flows in the avionics network – Backup configuration

For the full demand set, the mean reconfiguration time is less than 2.5 seconds. However, for some switch failures, up to 10 seconds are required. This is because the traffic in the demand set has not been equally distributed.

Yet it should be noted that traffic continues to flow uninterrupted using either standard or backup path. The node- and link-disjoint formulation of configuration templates ensures this. Consequently, the only concern would be a disruption of this path while the configuration is ongoing. While this scenario is extremely unlikely for failures due to natural causes, appropriate measures such as advanced intrusion detection systems need to be taken to ensure that such an intentional failure is equally unlikely.

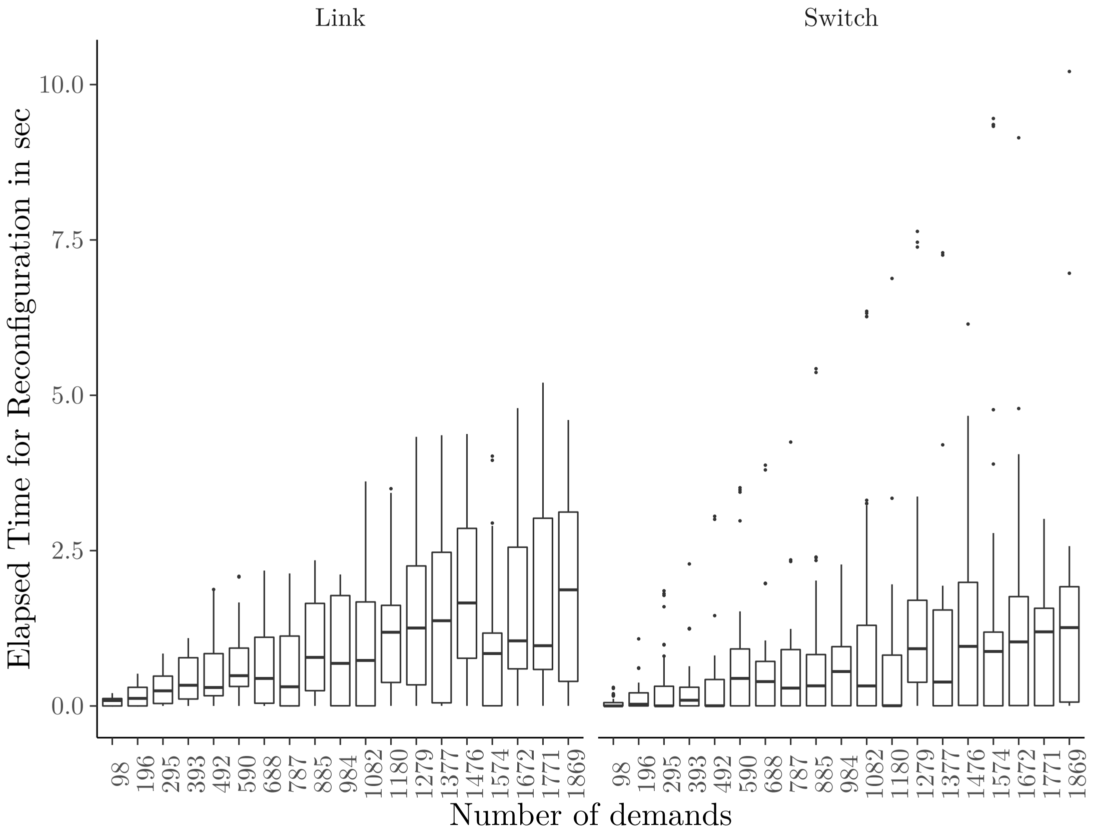


Figure 5.6: Time required to recover from failure in the avionics network

5.1.5 Impact of Successive Failures

In addition, the impact of several successive failures has been studied. Here, figure 5.7 shows how many flows are affected by a failure, depending on how many previous failures have occurred. The distribution remains constant for the first few failures ($n \leq 7$), then drops for $n > 7 \leq 12$ and then rises again. The first few failures do not have a significant impact on restored paths. However, this is because switch failures cause the isolation of hosts directly connected to that switch. Hence, as the number of failures increases, the number of affected flows decreases as the demands of isolated hosts are already disconnected the network. Finally, as even more failures affect the paths of the remaining flows, these become increasingly impacted.

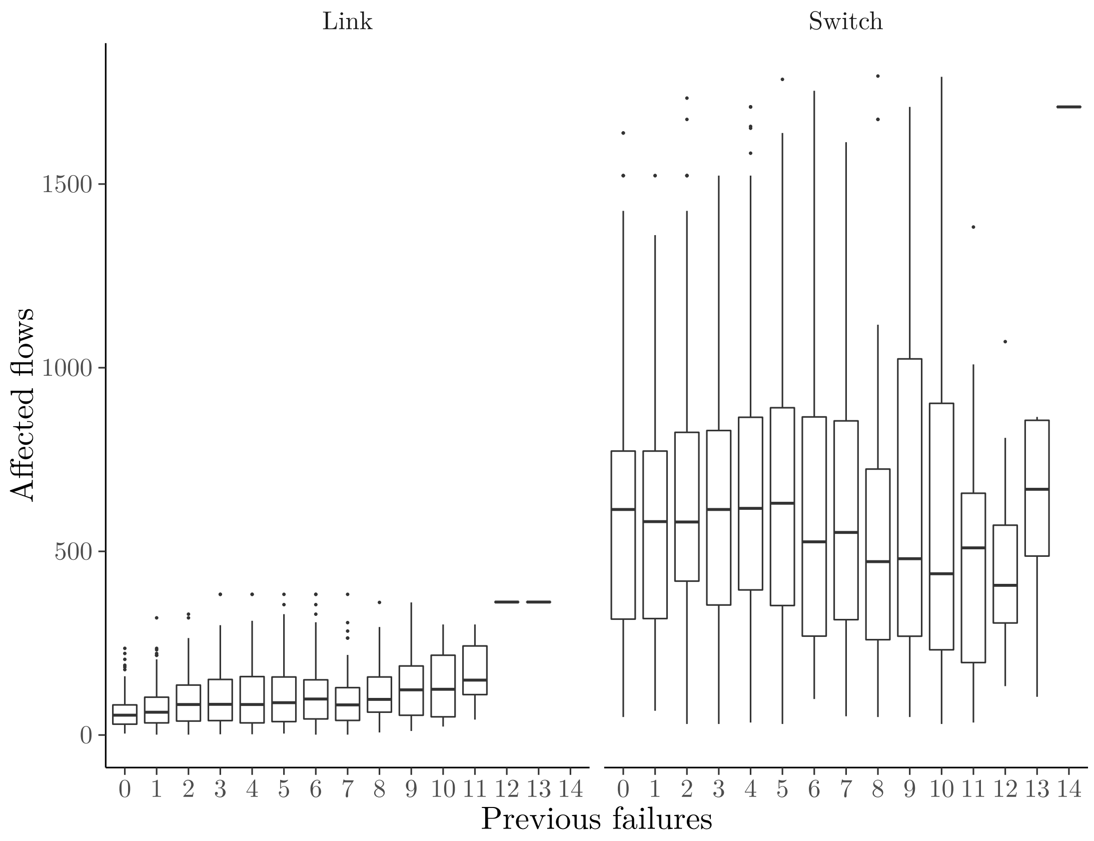


Figure 5.7: Number of flows affected failures in the avionics network

Success

To establish the effect of a failure, it is also relevant to consider how many flows cannot be recovered because the host is isolated. This isolation can either occur due to the failure of the switch to which it is connected or by the successive failure of all links that connect the switch to the network. This is depicted in figure 5.8 for the standard path and figure 5.9 for the backup path. In both cases, with increasing previous failures, the number of flows affected by a link failure increases, while the number for flows affected by a switch failure decreases. This is because link failures affect available backup paths, while switch failures affect the demand overall as the demand cannot be incorporated into the network. Consequently, the number of demands decreases over time, which in turn reduces the number of demands that are affected by a previous failure.

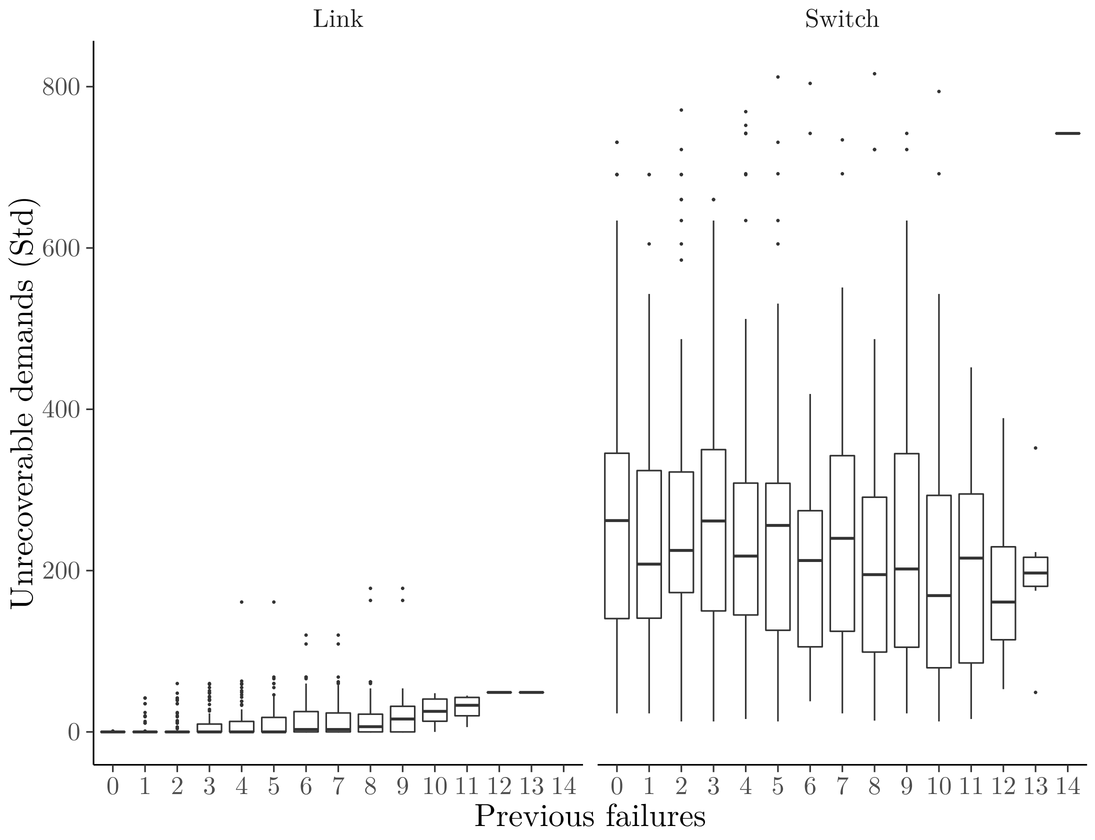


Figure 5.8: Number of unrecoverable flows with previous failures in the avionics network-Standard configuration

Timing

As depicted in figure 5.10, the recovery time varies with the number of previous failures. As the number of demands reduces with preceding failures, the time needed to reconfigure the network also reduces. For the avionics network, previous failures do not increase the search time for a path unaffected by those failures measurably. Notwithstanding, especially for a larger scale network such as the electricity grid – where a plethora of options may be available – this can be a significant factor.

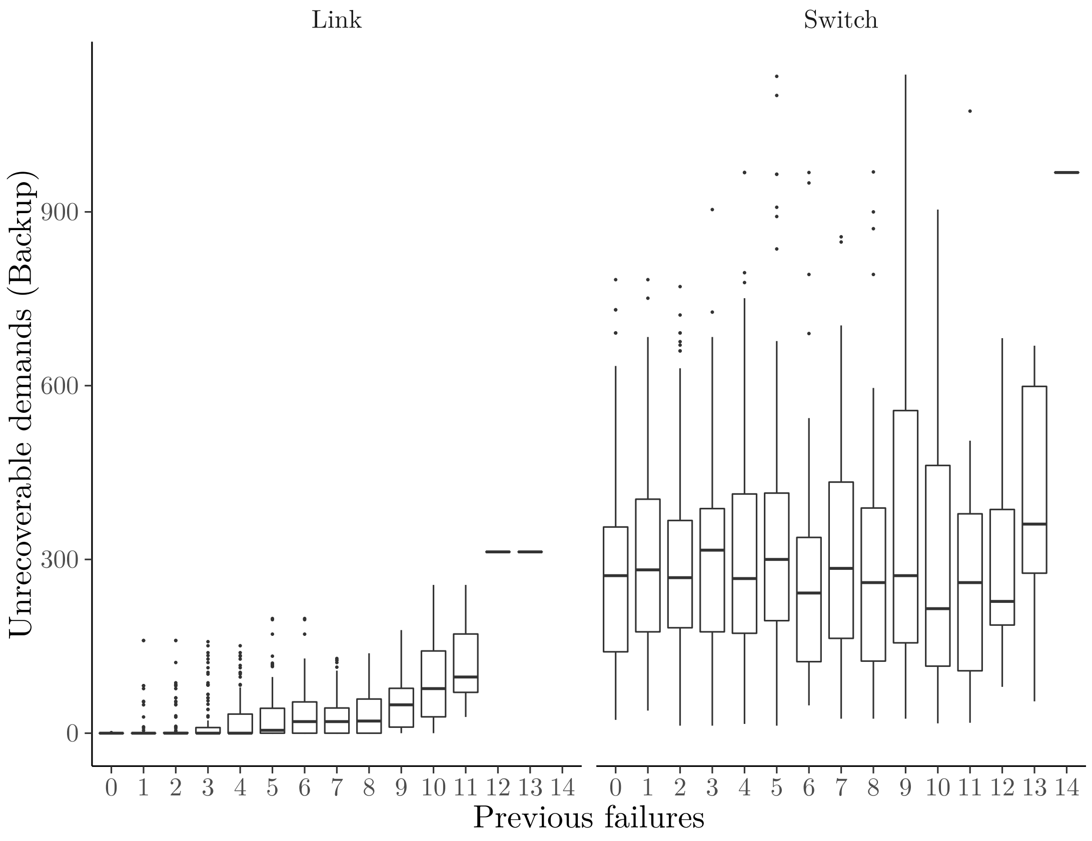


Figure 5.9: Number of unrecoverable flows with previous failures in the avionics network-Backup configuration

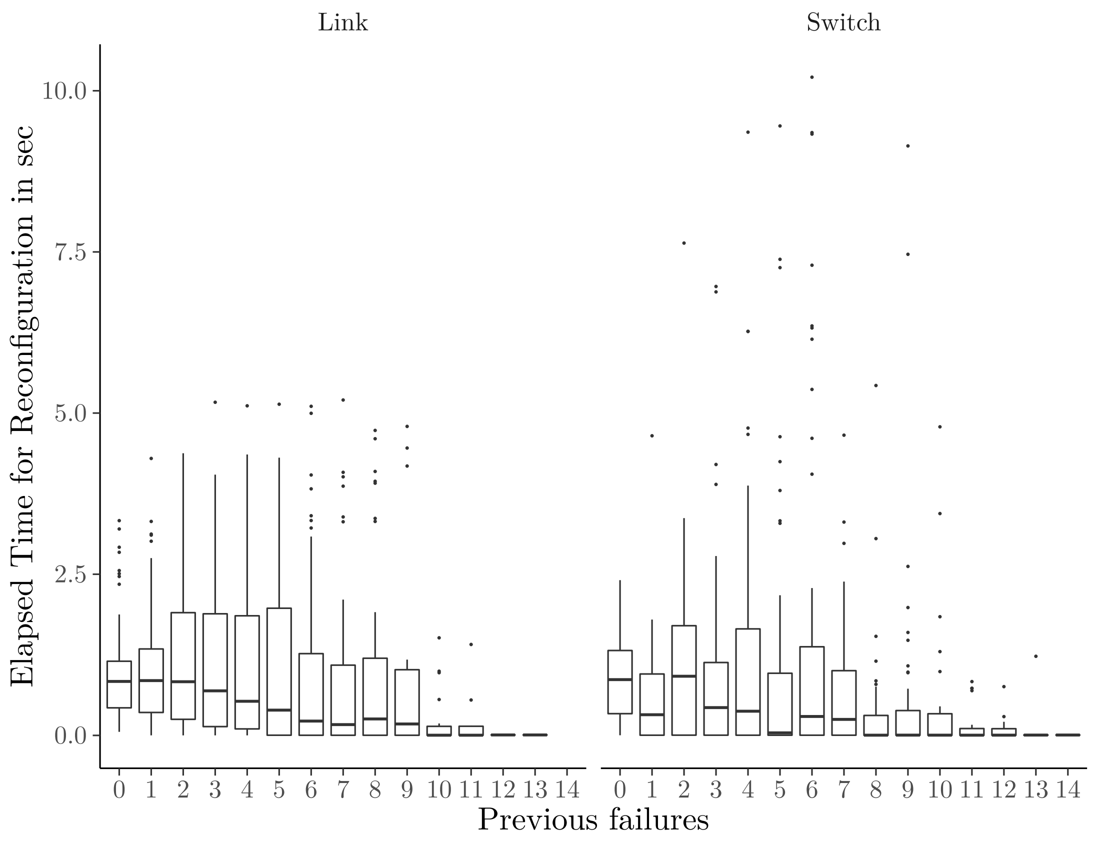


Figure 5.10: Time required to recover from failure in the avionics network

5.1.6 In-band Versus Out-of-Band Control Network

The analysis of the forwarding rules detailed in the previous section already provided some insight into the amount of command traffic that needs to be sent. Yet in order to truly analyse the impact of using SDN in safety-critical systems, the impact of using in-band versus out-of-band traffic needs to be considered. On the one hand, by using an in-band control network conflicts between control traffic and payload could arise. On the other hand, a separate control network increases the weight and complexity of a system. Consequently, it is relevant to analyse the impact of the presented solution on the control plane in addition to the impact on the data plane.

Strategies such as VLANs are available to ensure that the amount of control traffic is limited. However, it should be noted that such methods could increase the time needed the reconfigure the network. Consequently, the time span during which an additional failure could potentially not be tolerated increases. Yet this could be mitigated by increasing the number of redundant paths that are served concurrently for the most critical demands.

Moreover, the (expected) number of control traffic also needs to be considered for the design and optimisation of network traffic and included in the optimisation problem as additional demands. While this would lead to an overdimensioned network, for safety-critical networks such as the avionics network it is necessary to ensure that the safety-critical messages on the data plane are not negatively affected.

In addition, a separate control network would significantly increase complexity and weight, with the former being undesirable for safety-critical systems and the latter for mobile systems. Beyond that, for an in-band network, the resilience of the data plane has already been verified if the *max. resilience* function is used.

Deleting All Rules Prior to Rerouting

As said, the first option would be to delete all forwarding rules prior to deploying new ones. In OpenFlow, there is no difference between the process of adding and deleting forwarding rules. Consequently, the same considerations as for installing forwarding rules apply. Obviously, deleting all forwarding rules beforehand ensures that only the current and relevant forwarding rules are installed. This is especially relevant for those systems that are dynamic and where the TCAM is used close to capacity. Now obviously, deleting all forwarding rules beforehand increases the time needed for reconfiguration, as the timing analyses above only considered the installation of new forwarding rules, not the removal.

As depicted in table 5.1, for the avionics network, very little reconfiguration is necessary. For the backup path, only one intermediate flow needs to be deleted in the average case. In addition, the rule at the switch linking the network to the target resource needs to be changed. For $n = 1869$, only about two percent of flows require two intermediate rules to be deleted. Yet this number is significantly higher if obsolete forwarding rules from remaining paths (i.e. neither primary nor backup) need to be deleted. However, for the avionics network, only 17.62% of flows have more than one path remaining for reconfiguration. This is due to the network topology and the requirement that resilient paths must be node- and link-disjoint. Moreover, as shown previously, the number of demands that need to be reconfigured decreases with an increasing number of failures.

Table 5.1: Number of Forwarding Table Entries to be Deleted

Configuration	**Min.**	**Median**	**Mean**	**Max.**
Backup	2	2	2.019	3
Remainder	2	3	3.027	6

Selectively Deleting

In contrast, as all paths are node- and link-disjoint, the modifications as performed in the experiments above would suffice to ensure that traffic is no longer routed to the failed component after the entries in the first and last switch of the path has been modified. Consequently, the time needed to reconfigure the network is reduced or stays the same if there is a direct link between source and destination switch. While the amount of control traffic does not change, the removal of obsolete rules can be delayed so that only a minimum amount of bandwidth is consumed at any given time.

As seen in table 5.1, for the avionics network, this would only reduce the amount of the control traffic by a third for paths not included in the initial configuration, as the mean path length is two for $n = 1869$ demands. In addition, unused rules could also be a security risk if an attacker uses now unused paths to introduce malicious traffic. Furthermore, in large/dynamic networks this approach would not be practical, as forwarding tables are limited in size. Consequently, flow tables should be cleaned up to ensure that there is space for introducing new flows or reconfiguration.

Notwithstanding, it is possible to delay the deletion of intermediate forwarding rules, especially for in-band control networks. However, this strategy only makes a difference if there is at least one switch between source and destination.

5.1.7 Control Traffic

According to the specification [133], each OpenFlow message contains a 32 Bit header, followed by the payload as follows:

- 32 Bit ingress port address
- 48 Bit OpenFlow Ethernet source address
- 48 Bit OpenFlow Ethernet destination address
- 16 Bit Ethernet type
- 8 Bit Protocol number
- 32 Bit IPv4 source address
- 32 Bit IPv4 destination address
- 128 Bit IPv6 source address
- 128 Bit IPv6 destination address
- 13 Bit VLAN ID
- 16 Bit TCP source address
- 16 Bit TCP destination address
- 16 Bit UDP source address
- 16 Bit UDP destination address

Consequently, each modification/deletion requires 581 Bit of traffic that needs to be exchanged. For the avionics network with $n = 1869$, up to 991 rules need to be modified/added for a link failure and 2286 rules for a switch failure. Thus, up to 575 kb of control traffic need to be transmitted for a link failure and up to 1.32 Mb for a switch failure.

For the avionics network, this amount of control traffic does not warrant an additional control network. Furthermore, this network is the on-board network of an aircraft. Reducing weight and complexity is one of the chief goals in aircraft design, hence an additional network would only be warranted if it would significantly increase safety. As the low bandwidth required would not significantly interfere with safety-critical traffic (especially if prioritised accordingly), it is not necessary in this case. Notwithstanding, this bandwidth requirement will need when generating optimisation templates.

This chapter considers a sample network on-board an aircraft. Nevertheless, especially for large-scale networks such as the smart grid with significantly more nodes, the control traffic might be significant. In those cases, it would be possible to limit the control traffic to a certain bandwidth to limit the queueing delay of safety-critical traffic.

Moreover, from a security perspective, it needs to be considered that a common transmission medium exists. Consequently, network level attacks such as DOS or replay attacks affect both data and control plane. Thus, in such attack scenarios, a controller could not post a modify-rule message to cut an attacker from the network. However, in safety-critical systems, one could consider other strategies such as only allowing modifications e.g. while the vehicle is parked. Notwithstanding, it needs to be included in the analysis.

5.1.8 Strategies to Delete Forwarding Rules

Beyond considering approaches to add forwarding rules, it is also necessary to investigate how to delete the rules that became obsolete as a component has failed. As mentioned before, the message size is identical for modification and deletion. In addition to reducing the bandwidth on the network, as the TCAM memory is limited, deleting obsolete forwarding rules is especially relevant for networks with a large number of flows.

In principle there are two strategies that can be employed: either deleting all obsolete rules prior to deploying the backup paths or selectively rerouting. The advantages and disadvantages of these strategies will be discussed in the following paragraphs. As the shortest path has been chosen as the primary path, only one forwarding rule needs to be deleted/changed (at the switch linking the target to the network). Consequently, differences in strategy will only matter for backup paths.

5.2 Management Plane Configuration

After discussing how a reaction can be performed in the data plane and analysing the impact on the control plane, this section describes the impact on the management plane.

The following considerations have been split into two parts: safety section 5.3 and security section 5.4. This has been done to consider resilience and accountability separately from issues such as data privacy. For safety-critical systems, security is one of several means to ensure safety.

The selected reaction needs to be *appropriate* to the impact on the network. As described in section 3.2, the impact of the mitigation effort affects data plane, control plane as well as management plane. However, as the decision as to whether a reaction is triggered is taken in the management plane, it will be discussed here.

In the previous section, reactions to failed network components were discussed. Here, a network reconfiguration has been deemed necessary so that safety-critical traffic can reach its destination. Consequently, the controller can directly trigger the reconfiguration without interaction with the management plane. This decreases the response time and consequently reduces the chance that there is a second failure during the reconfiguration.

However, in the general case, it cannot be assumed that the impact of the anomaly and a potential reaction is as straightforward to establish. In addition, malicious devices could falsely report failure of other devices to cause unnecessary reconfiguration or ensure that traffic is routed via them. Consequently, a more complex design of the management plane is required. The overall system layout is shown in figure 5.11. As depicted, there are three types of equipment that serve as sensors: IVHM, SDN, and IDS. A detailed description of the data thus available has already been given (cf. chapter 2).

However, it should be noted that not only complete failures can be detected by the SDN controller but also issues such as an increase in the queueing delay at an interface. In addition, reports by the health management system or intrusion detection are collected in the management plane and could trigger a reconfiguration or other reaction. These sensors provide the input to the 'controller' of the closed-loop control system as depicted in figure 3.2. This functionality of the management plane again consists of three functions: anomaly detection, reaction finding and reaction implementation. Here, the anomaly detection component decides whether the event reported by one of the sensors is indeed an anomaly. Subsequently, the reaction finding process determines whether a reaction is needed and if so, what it should be. Finally, the reaction implementation component interacts with the control plane to trigger the execution of the determined interaction. The functionality of the latter two components will be described in more detail below.

5.2.1 Finding an Appropriate Reaction

Once an anomaly has been detected, the next step is establishing a suitable reaction to mitigate the effects of the anomaly. In section 3.2, the concept of *appropriateness* has been introduced. As mentioned, the impact of the anomaly is compared to the potential impact of a configuration change. If there is no significant positive benefit, no reaction should be performed. In addition, if the reaction to a purely security-related incident would have a negative impact on a safety-critical functionality, the reaction should not

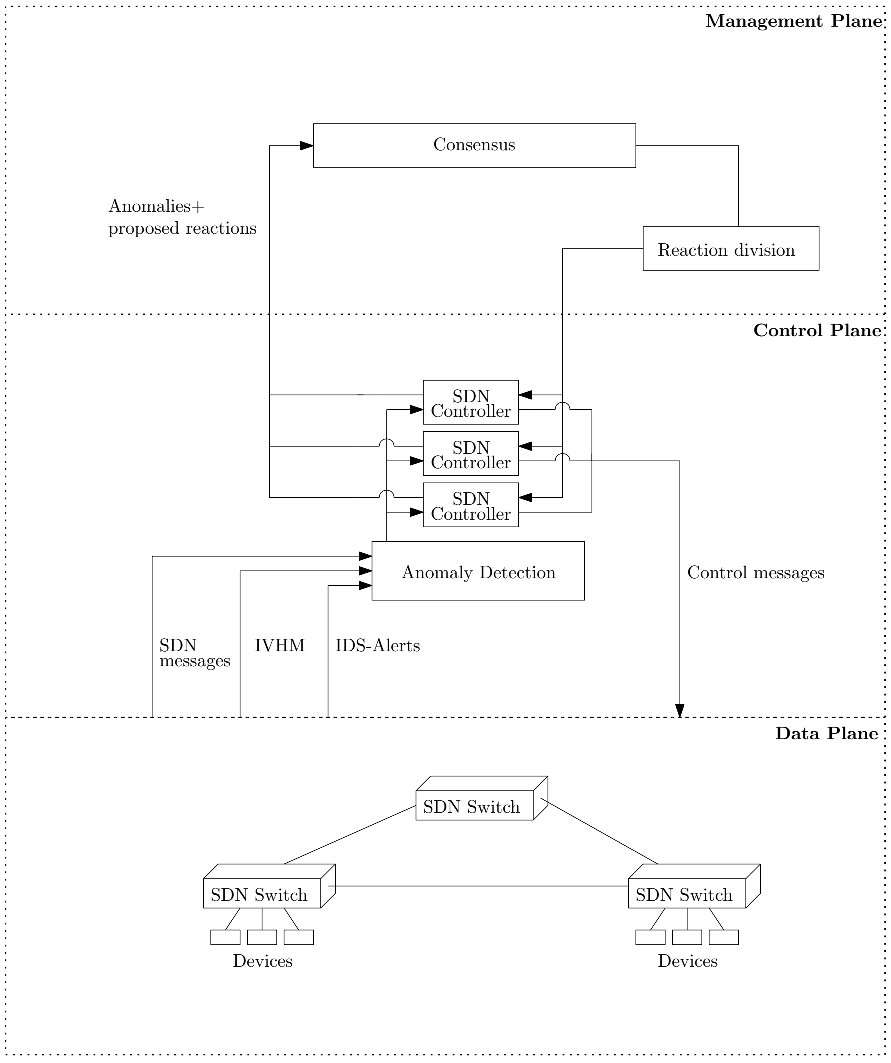


Figure 5.11: Management plane configuration

be implemented. Moreover, it may be the case that no response can be performed online, because a given anomaly can only be mitigated offline, e.g. by physically replacing the component or patching software. Thus, degraded modes are by no means obsolete using the mechanism described henceforth, but they serve as an additional level to increase the availability of safety-critical functionality.

For the safety-critical networks studied in this thesis, the following characteristics determine whether a proposed reaction is appropriate:

- impact of anomaly on network performance (throughput, latency...)
- impact of reaction on network performance

- amount of flows that need to be rerouted
- critical traffic that is affected
- false positives/negatives

First of all, it is necessary to establish whether the failure had any measurable impact on network performance. If this is not the case, either the failure affected a component that did not handle any traffic, or it has been a false positive. In both cases, no reconfiguration should be performed.

Secondly, the (at this point) theoretical impact of the suggested configuration needs to be established. Here, the potential benefit of the network reconfiguration such as lower link usage, rerouting around a failed component has to be balanced against the impact of a network reconfiguration such as number of forwarding tables that need to be changed. In addition, the possibility of false positives (see below) needs to be taken into account.

Thirdly, the amount of flows that need to be rerouted is of importance. A slightly increased buffer size does not necessarily call for a reconfiguration that affects a significant proportion of flows. In this context, it is required to consider that the installation of new forwarding tables takes time. Thus, another, more significant anomaly cannot be addressed while this network configuration takes place. Consequently, the number of flows that need to be rerouted and the number of forwarding table entries that need to be changed is significant.

Fourthly, the amount of critical traffic that is affected needs to be considered. If only less critical traffic (e.g. maintenance data) is affected, a network reconfiguration may not be necessary. On the other hand, even if just a few demands of most critical traffic is affected, a reconfiguration that rectifies this issue may be called for.

Finally, no matter how thorough the system design, both false positives and false negatives can occur. Consequently, steps need to be taken to ensure that neither negatively influence safety. Here, it is especially relevant that false positives – whether they are benign or caused by a malicious controller – do not result in network reconfigurations that negatively affect critical flows.

All these parameters have an impact on whether the reaction is appropriate. Any event can result in either a change within the network state or cause a state transition to occur. For anomalies that results in a state transition, the new reaction set is used e.g if security a security incident has been detected:

$$s_1 \rightarrow s_2 : \binom{n_2}{1} \in \mathcal{M}_1 + \mathcal{M}_2 \subseteq \mathcal{M} \tag{5.1}$$

For the safety-critical systems considered in this thesis, the state transitions with respect to safety can be clearly defined. Here, any violation of the requirements defined in the optimisation problem results in a state transition from s_1 to s_3 or s_4 i.e. from safe and secure to not safe and (not) secure.

In addition, the traffic patterns of safety-critical systems are generally well-defined and do not need to consider dramatic variations in traffic patterns, as opposed to Internet services. Consequently, it is comparatively easy to detect gradual changes that are indicative of an emerging problem, if not already detected by IVHM mechanisms. Moreover, it should be noted that the theoretical latency distributions used to obtain configuration templates (i.e. by using the *maximum resilience* objective function) are worst-case assumptions with all resilient paths being served concurrently. Consequently, if the *measured* latency distribution and resource utilisation reaches this worst case value, corrective actions should already be considered to ensure that no safety-critical increase in delay and utilisation is reached. To this end, a suitable reaction from $\mathcal{M}_1$ may be selected, as the network is still in s_1 at this time.

Notwithstanding, it should be noted that other, unrelated failures (such as engine failures) could also result in an increase or change of traffic. Obviously, such emergency states need to be considered when designing anomaly detection, to avoid aggravating the safety state of the system.

With respect to security, a similar approach may be taken. However, unlike the well-defined, hard transitions that apply with safety (a missed deadline is a missed deadline), the state transitions with respect to security are more more difficult to define. This is on the one hand caused by an uncertainty in the sensors, such as intrusion detection systems. On the other hand, a loss in confidentiality cannot be easily quantified. Here, it is necessary to link outputs of ID systems to the confidence of the output. Here, it is beneficial that the sensors are being applied to safety-critical systems. This allows to have far more detailed information on quality indicators such as precision and recall. Consequently, those can be used to add the respective uncertainty, which in turn allows to judge the impact of the detected security anomaly. This is especially relevant when resilient systems establish a different conclusion on the network state, such as different SDN controllers.

After establishing the impact of the detected anomaly on the network for the safety and security parameters as given in chapter 3, the current system state at time t can be established as $s(t) \in S, S = s_1, s_2, s_3, s_4$. Subsequently, the available reactions need to be assigned to the corresponding network states. The following actions are sample reactions that could be available in an avionics network as employed for the network reconfiguration in section 5.1, assigned to the according mitigation set m, as detailed in equation (3.8):

- m_1
  - Doing nothing
  - Log incident
  - Report item for maintenance/trigger maintenance action
- m_2
  - Temporarily increasing packet inspection rate/adapt firewall
  - Permanently increasing packet inspection rate/adapt firewall
- m_3
  - Reconfigure network
  - Enter degraded mode
  - Alert operator
- m_4
  - Block traffic
  - Isolate parts of the network

If the network is safe and secure (i.e. in s_1), the first possible reaction is to do nothing. This reaction is suitable for transient anomalies that are caused by external events, such as individual bit flips due to the effects of solar radiation. These anomalies will not have a negative effect on the system, and hence do not need to be addressed. If this is not sufficient, a log of the incident can be generated and stored for further analysis. Depending on available systems, the frequency of occurrence, and the information associated, this report can either be periodically offloaded or immediately transmitted to a supervising entity (e.g. an operations centre). Moreover, if a component is defective but safety not affected (e.g. because it is one of three redundant systems), the item can also be reported for maintenance and/or an according maintenance action triggered.

This option is also available to mitigate minor security vulnerabilities that do not have a detrimental impact on safety.

If the network is no longer secure but still safe, reactions with a more profound impact become available. If a loss of confidentiality or integrity is likely, the frequency of inspections from intrusion detection systems may be increased. As this results in delays of the traffic, these measures can be introduced either temporarily or (if more severe) permanently. Together with logging (as available in m_1), this could provide a more detailed insight into an intrusion, and thus help to patch relevant vulnerabilities. In addition, firewall rules (if present) could be adapted to limit or block external communication, if this communication link is not safety-relevant. For example, in an aircraft, passenger internet could be blocked, while crew links to air traffic control need to remain intact.

Going beyond this, if the system is no longer safe or if safety margins are significantly reduced so that a loss of safety is imminent (e.g. if two out of three redundant components fail), the reactions of set m_3 become available, in addition to m_1 and m_2. In this reaction set, the network reconfiguration as detailed in the previous section are included. While ongoing transmissions are not negatively affected, a reconfiguration limits the number of resilient paths available for future reconfigurations. Hence, it should only be considered if the detected anomaly significantly reduced safety margins. Furthermore, this mitigation set also includes the option to enter a degraded mode, where only the most critical operations continue. In the aircraft example, this would be the functions related to flight control. Concurrently, it is generally necessary to alert an operator to the fact that the system went into a degraded mode. Depending on the system, this operator would be able to initiate further recovery protocols not deemed suitable for automatic execution.

Finally, if the unsafe state has been caused by a security incident, it is possible to block traffic or isolate parts of the network, to allow as much traffic as possible to be transmitted while containing the event. However, such an extreme mitigation should only be performed as a last resort, as it can have a significant impact on safety. As an additional safeguard, it could be considered that such an action is only performed after confirmation of an operator (if present), but without further manual action by them.

5.3 Safety Considerations

SDN features a *logically* centralised controller. However, that does not mean that there can only be one controller in the network, which would be a single cause of failure. Notwithstanding, it should be noted that the data plane continues to operate if a controller fails, but no changes to routing can be performed. False negatives are far less likely in the safety context (see section 5.4 for the security context), as benign failures likely result in some impact on the network state that may easily be detected. An exception here are intermittent failures, that may go undetected for a period of time. However, such failures e.g. radiation are generally self-correcting and do not need any high-level response, at least if sufficient resilience is available in the network.

5.3.1 Strategies for Controller Resilience

There are several possible strategies to allocate controllers to the network.

- Primary/secondary controller
- Subnetwork division (overlapping/non-overlapping)
- Consensus amongst controllers

Firstly, it is possible to have one primary controller that is responsible for the entire network and a secondary controller that is either in hot or cold standby to take over if the first controller fails. However, with this arrangement, only complete failures can be mitigated, while transient faults and security incidents are not considered. What is more, a suitable mechanism needs to be designed that can reliably detect when a handover from primary to secondary controller is necessary. Finally, with this configuration, an attacker could sequentially target the respective controllers to cause a system failure.

Secondly, the network can be subdivided into different subnetworks. Here, both scenarios where the network boundaries overlap and where they do not can be envisaged. In both cases, due care needs to be taken with system design so that critical components are distributed across the system in a way that prevents loss of critical traffic with any one controller failure. With overlapping subnetworks, more resilience can be provided against non-malicious failures, as the second controller can seamlessly overtake failures of the first. What is more, this strategy of overlapping networks is already used in some critical systems, e.g. the hydraulic networks of aircraft, where control surfaces are supplied by at least two different hydraulic lines.

The third option is technically only a variant of the second. Here, several controllers are responsible for the network and need to come to a conclusion together. Consequently, such a configuration would offer Byzantine fault tolerance, as a majority of controllers needs to be corrupted to prevent correct system operation from a security perspective. From a safety perspective, $n-1$ controller failures can be tolerated.

5.3.2 Time Required for Consensus

The system proposed in this thesis introduces an extra checking feature, to ensure that the reconfiguration is safe. While this process does not negatively influence the running configuration (since only either standard or backup path are rerouted), another failure during this process cannot be tolerated and could result in traffic loss. Consequently, there is a balance between the time required for the reconfiguration and the thoroughness.

5.3.3 Accountability

While previous work (cf. section 2.3) covered some strategies to coordinate different controllers, one key feature has not been addressed: accountability.

Just as every current maintenance effort in a safety-critical system has to be logged, all controller actions on the data plane would need to be accounted for. This permits to detect both rogue controllers and any potential problems with current configurations so that they can be rectified as soon as possible.

To this end, the following information needs to be collected:

- Which component detected an anomaly?
- Which controller has proposed which reaction (if any)?
- Which controller has implemented which reaction?
- Which switch reported that it has implemented a change to the forwarding tables?
- When did those events occur?
- Network state snapshots.

First of all it is important to record when an anomaly (of whatever kind) has been detected. Along with SDN controllers, other components such as IDSs and integrated health management components could also report anomalies. This record allows to see if the reaction is justified, up-to-date and useful. What is more, recording such anomalies allows to investigate possible correlations between different anomalies, that may require different responses if faults occur individually. Beyond that, it helps to avoid false positive reactions, if the anomaly does not have a measurable impact on the network.

Secondly, it needs to be recorded which controller has proposed a reaction. If one controller consistently produces disagreeing reactions, it might be indicative that the controller is defective or has been corrupted. Consequently, it is important that decisions are recorded to ensure that timely maintenance action can be performed to replace or patch the defective controller.

For the same reason, each implemented reaction needs to be recorded. Additionally, this record also allows to verify that no unauthorised configuration changes are implemented.

Beyond that, the switches report that they have successfully changed the content of their forwarding tables. If all switches report that the changes have been performed and the network state still does not improve, the implemented reaction did not bring the intended results. This means that either another reaction needs to be performed or that the configuration templates are no longer useful and hence need to be recalculated.

Adding a time stamp to all events further adds the possibility to correlate various events. In addition, if one controller detects an event significantly earlier or later, this could provide additional insights, especially for security events.

Finally, it is important that regular snapshots of the network state are recorded. As mentioned above, this allows to supervise various components and also helps with the decision making process. Furthermore, regularly monitoring the network state allows to detect a slowly degrading network state e.g. due to ageing components, that may not otherwise be found in time.

5.4 Security Considerations

5.4.1 Authorisation

To ensure that a malicious controller cannot negatively impact system operation, it is necessary that any changes to the network are authorised. As a single authorisation instance would face the security exposure risks as a single controller, a distributed approach is needed.

In this context, the multi-party authorisation (MPA) model can be used. Here, multiple parties need to review and authorise actions prior to their execution. Otherwise, the action is not permitted. To govern the authorisation process, either pre-formulated policies or spontaneous negotiations may be used. Common governing criteria include the number of authorisations required, and identity/role of specific authorisation entities.

A formal definition of MPA has been established in [108, sec. 2.3]. The following properties need to be met according to her definition by an authorisation protocol:

Agreement Honest entities agree on the content subject to the request.

Validity Honest entities process the request initiated without modifications.

Temporal order Authorisations are granted in the order defined.

Non-triviality The protocol is not permitted to abort the process.

In the context of SDN controllers in critical systems, this means that the controllers need to concur on the configuration template that is to be implemented. Furthermore, it needs to be ensured that no different configuration template is introduced in the process. Beyond that, the order of approvals needs to be followed. Finally, it is not allowed that the process terminates without obtaining a reaction to the detected anomaly. However, it is permissible that the outcome of the consensus is that no reaction is necessary, e.g. because only a minimal increase in the queueing delay has been observed.

Additional reactions, such as an increased logging frequency may also be implemented directly at the component. However, since they do not have an influence on the safety-critical traffic, participation in the consensus process is not necessary as such.

5.4.2 Privacy

Some critical systems mainly process technical data, hence data privacy concerns only play a minor role. Notwithstanding, privacy also needs to be considered in this context. Where data privacy is especially important (e.g. in the smart grid), additional means such as encryption need to be considered.

However, as data is persistently recorded, and safety-critical systems are in operation for significant periods of time, it may be the case that encryption means are becoming obsolete during the operation. With the increase of computational power or new algorithms, older encrypted entries may no longer be sufficiently protected. While some data may no longer be required to be private at this point, this effect should nevertheless be considered.

Consequently, collecting data needs to be kept to a minimum, especially considering the likeliness of false positives. On the other hand, it is important to record all the necessary data to be able to reproduce a security incident in as much detail as possible. As mentioned before, this allows to prevent the incident from having an impact again or to other systems. To prevent an attacker from interfering with such forensic details, it is necessary that they are recorded in a tamper-proof way. Notwithstanding, the accountability requirement for safety can be seen to be orthogonal to data privacy.

5.4.3 Attacks on the Controllers

While those elements delay the reaction to a detected anomaly, it is very important from a security point of view given that this system could also be exploited. The following attack paths can be envisioned. Firstly, a necessary reaction can be prevented from being executed by n malicious controllers.

Secondly, one controller can spoof the necessity to react to a failure, while no such failure has taken place. This can on the one hand result in the flooding of the reconfiguration interface by too many requests being sent. On the other hand, it can trigger a reconfiguration that is not necessary, thus degrading system performance, to the point where no more safe reconfigurations can take place.

5.4.4 Completeness

Furthermore, it may be the case that a security issue is correctly detected, but the proposed reaction would negatively impact safety-critical systems. In those cases, an attack may have to be tolerated. An example for such a case is a minor security incident that would require significant reconfigurations, comparable to the scenario written above.

Additionally, a security incident may not be completely mitigated during operation, because the function is provided by the affected component which is safety-critical. Here, it may be necessary to tolerate an attack for a given period of time, until either a patch can be provided or the affected component can be replaced. While component diversity should reduce the chance of several safety-critical components being affected by a critical vulnerability at the same time, previous experience such as with Stuxnet shows that it needs to be considered.

Beyond that, as mentioned above, both false negatives and false positives of IDS can occur. From a security perspective, false negatives are especially concerning. Unlike e.g. flooding, which can be detected on a network performance level, more sophisticated attacks may not necessarily effect the network in a way that can easily be detected. Consequently, a malicious entity may prepare a more sophisticated attack. However, once an attack has been launched, it can be detected from a system level and hence mitigated, if the safety critical functionalities are affected. If this is no longer possible, it is necessary to resort to degraded modes.

5.4.5 Integrity

Message integrity itself can be ensured using signing mechanisms already included (albeit optional) in SDN. However, replay attacks of previous configuration changes need to be prevented. Especially when further failures have occurred, configuring based on a previous configuration template can have a detrimental effect on safety.

5.4.6 Network Attacks

Beyond those aspects that are of specific relevance to SDN, it should also be noted that most common network attacks can also be relevant. Consequently, the methods such as encryption need still be used in SDN networks. In addition, fingerprint attacks and DOS have also been performed successfully in SDN. Further details on possible attacks and mitigations can be obtained from [157].

5.5 Summary

After describing the experimental setup, this chapter has analysed the impact of dynamic network reconfiguration on the control plane. It has been shown that for the AFDX network, switch failures have a far more severe impact on the network than link failures. As devices that are originators of demands are attached to only one switch, a failure thereof obviously results in the loss of the demand, with is not the case for link failures.

Moreover, the impact on the data plane for in-band control networks has been discussed. It has been found, that for the AFDX network, an in-band control network would not have a detrimental effect on safety-critical traffic. Finally, methods to delete forwarding rules prior to installing new forwarding rules have been discussed.

Furthermore, the necessary considerations for control plane resilience have been presented. Considerations for safety and security were discussed.

The impact on safety, resilience, accountability and appropriateness of a reaction were detailed. Focusing on appropriateness of a reaction, a model that matches the expected impact of a reconfiguration to the network state has been presented. This allows to reduce unnecessary severe reactions to minor issues.

Beyond that, issues related to the security of the system have been discussed. Here, maliciously acting controllers, data privacy and the completeness of an implemented reaction have been considered. The latter also includes considerations on the accuracy of both the detection and mitigation of an attack.

Chapter 6

Discussion

This chapter discusses the approach taken in this thesis, and is divided into four parts. Section 6.1 considers general points, while section 6.2, section 6.3 and section 6.4 discuss issues related to the data, control and management planes, respectively.

6.1 General

This section discusses the general approach followed in this thesis, which can be further subdivided into theoretical and practical considerations.

6.1.1 Theoretical

To begin with, process, the model used and the algorithms are considered.

Non-Deterministic Process in a Safety-Critical System

One of the most serious impacts of a dynamic reconfiguration is on determinism, which has been an important criterion for safety certification. While predictable processes/reactions are sufficient to address safety incidents, this does not apply to security incidents. If processes are deterministic, an intruder could establish this fairly quickly and change their behaviour accordingly. Consequently, the designed reactions will no longer sufficiently protect the network. In order to prevent this, the selection of an appropriate reaction must include a certain element of randomness, and hence being no longer deterministic.

By dividing the set of possible reactions into four parts that correspond to the network state, the process described in this thesis ensures that the impact of the reaction is proportional to that of the incident. This assures that the reaction does not have a more detrimental effect on the system than the original incident. Furthermore, it still offers some protection against intruders, as they cannot be certain which reaction will be selected by the system. As the response set increases depending on the impact on the system. Hence, the more serious the effects of the incident, the harder it gets for an attacker to predict the reaction.

In addition, if the reactions are not hard-coded, the system may also address incidents (both safety and security) that were not foreseen at the design stage. While testing in safety-critical systems is far more stringent than in other commercial IT systems, unexpected failures may still occur. Consequently, a system that focuses on the network state (as described in this thesis) rather than individual events is more likely able to continue its operation without relying exclusively on human operators.

Notwithstanding, while the reconfiguration is in progress and until the switches have reported successful implementation to the controller, the state of the flow tables is not defined. Consequently, it needs to be ensured that the complete path has been configured before the first packet is sent on its way.

For this, several approaches exist. On the one hand, the network could default to dropping all traffic that does not have a rule associated. However, this would prevent the identification of new devices, which is not desired in mobile networks or the smart grid, where new users are added continuously. On the other hand, unnecessary control traffic will be broadcast if switches have to ask the controller(s) how a flow should be handled while the reconfiguration occurs. While the delay experienced by the arriving flow is not critical (because it is already served by another, node- and link- disjoint path) the control channel could become saturated or critical traffic could be delayed or dropped due to insufficient bandwidth (for in-band control networks). Consequently, the advantages and disadvantages of the reconfiguration strategies need to be carefully considered and tailored to the requirements of the individual system.

In the context of the avionics network presented in this thesis, new devices will only be added when the aircraft is being maintained (i.e. on the ground), as most avionics bays are hard to access during flight. Consequently, while the aircraft is operating, dropping all unidentified traffic by default is preferable from a security context. Notwithstanding, such events need to be logged, so that they can be analysed later on.

Hence, a network reconfiguration would only be triggered in response to a detected component failure or attack. As the configuration templates fulfil all requirements of safety-critical traffic, if a reaction is deemed to be *appropriate* (see below), the reconfiguration is guaranteed not to interfere with critical demands.

While the selection process itself is deterministic, the reconfiguration is not, as the selected alternate paths not only depend on the last failure, but also on any previous ones. Consequently, the process is not deterministic. This can be seen as problematic in a safety-critical system, as the state transitions are not pre-defined.

However, all configuration templates are safe and also consider the worst case scenario with respect to the delay bound. As the reconfiguration process ensures that two node- and link-disjoint paths are always available, at least one path continues to route traffic without being interrupted. Consequently, while the reconfiguration is not deterministic, no safety-critical traffic is negatively affected.

Notwithstanding, a conflict between safety (which requires determinism) and security (which cannot be too predictable lest an intruder may exploit the very fact) remains.

6.1.2 Model

Another important aspect of this thesis is how the investigated safety-critical systems have been modelled. In this thesis, the reconfiguration process has been modelled as a simple control system problem, where both attacks and (unintentional) failures are considered to be simple disturbances. While it is a standard way to deal with failures, the part of an attacker is simplified by this approach, as it does not reflect the active participation and active interest to cause a failure.

While game theory is commonly used in the security domain to address the interplay between an attacker and a defender, this approach cannot be used in this context. In safety-critical systems, the attacker and defender abide by different rules as the defender may have to tolerate an attack if a reaction could compromise a safety-critical functionality. What is more, game theory does not address *how* the players play. An attacker follows only a subset of rules of the defender, since the latter is constrained by the safety-critical functionality that must not be affected. Consequently, a game-theoretic approach is not suitable as different sets of rules for either player cannot be incorporated in the model.

Moreover, in most cases it is hard to distinguish between intentional and unintentional failures. On a network level, only the *effect* of either can be observed. Consequently, a game-theoretic model considering only an interplay between attacker and defender would also be an insufficient representation, as it cannot address failures that have not been caused intentionally as there is no 'benefit' for either player in this case.

Hence, the model used in this thesis strikes a balance between modelling safety and security incidents by focussing on the impact of the system rather than on the cause of the disturbance.

Algorithms & Functions

Another theoretical aspect that needs to be considered are the algorithms and (objective) functions used to obtain configuration templates. This work only considers basic optimisation functions and heuristics, and not approaches specifically designed to a given critical system. However, individual critical systems will differ significantly from each other and consequently the focus of the optimisation functions will differ.

Moreover, even for critical systems, published algorithms generally improve one aspect of the basic algorithm – such as Dijkstra's – to adapt it to their specific needs. Thus, the comprehensive comparison and effects of algorithms facilitates the selection of a suitable algorithm for a specific application. By focusing on basic properties that are relevant to safety-critical systems, choosing an algorithm and then tailoring it to a specific application is facilitated compared to using a specific algorithm from literature that has already been tailored to another application.

With respect to the optimisation problem, this thesis additionally provides a general set of constraints that can be applied to a specific system. If needed, minor modifications such as different resilience requirements for different subsets of demands may be easily made.

6.1.3 Practical Considerations

Beyond those theoretical points, there are also practical issues that need to be considered.

SDN Technology

To begin with, the used technology (SDN) should be analysed. Here, a potential issue is that SDN is a fairly new technology. Consequently, failure statistics and reliability data lack historical data. Hence, they may not be representative for a critical system that will be in operation for several decades.

In addition, it cannot be guaranteed that the shortcomings described for current systems (difficulty to source replacement parts, obsolete technology) will not affect SDN to the same degree. While new(er) technologies such as P4 can also be used to implement SDN and previous versions of the OpenFlow standard have been downward compatible, no guarantees exist that this will be the case for the operation scope of most critical systems.

However, current systems face all those shortcomings and in addition do not allow to react to detected failures. Yet the ability to react to detected security incidents has been identified as an important measure to deal with an increasing exposure to cyber risk. Consequently, continuous use of current systems is only appropriated in legacy systems that cannot easily be updated to include reactive technology. However, it is still important that those systems are protected by passive measures to reduce their vulnerability.

Capacity

Another issue is that the system presented in this thesis requires significant spare resources in order to provide additional resilience. While it can be added during operation when spare resources exist, it will often have to be considered in the design stage. This is due to the effect on resource usage if more demands have to take a specific route to address a failure. However, the model does not incorporate methods that facilitate design stage considerations, but rather needs to be iteratively integrated in the design stage.

Yet, as safety-critical systems tend to be built upon gradually, this issue is less problematic here than for more dynamic, non-critical systems. Even new critical systems tend to be based on prior designs. In addition, while a system may not be designed a priori for adaptive network management, the mechanisms presented in this thesis may still be used to increase network resilience.

In addition, as seen for the avionics network, safety-critical systems tend to be designed with significant spare resources. As future operational requirements cannot be easily anticipated and replacing hardware is generally more expensive that using more capable links, such design approaches are commonly used for critical systems. Consequently, this spare capacity can be used to address failures initially. Notwithstanding, it should be considered that adding further traffic can also reduce the ability of the network to tolerate failures due to capacity constraints.

6.1.4 Number of Failures Tolerated

This work has introduced a system that can address up to ten successive component failures, caused by both intentional and unintentional events. Moreover, it provides a mechanism to find an appropriate reaction to other events (such as unexpected network flows). While extends previous work by covering both failures and attacks, a key problem remains: only a limited number of failures can be tolerated before some demands can no longer be rerouted and performance degradation has to be accepted.

Regardless of complexity, no network will be able to tolerate an unlimited amount of failures. The mechanism presented in this thesis ensures that the network is being kept operational until suitable repairs can be carried out. This would e.g. allow a system to be kept operational until a suitable maintenance facility is available. Hence, a vehicle could continue its journey to its intended destination or a part of the smart grid could wait until the power plant is scheduled for maintenance rather than ceasing to supply parts of the city. Likewise, it would prevent the need of an emergency landing of an aircraft thus allowing it to continue its journey an airport where the spare parts are available.

Such a capability would be especially relevant for security incidents, where systems could safely continue operation until a safe software release is certified.

6.1.5 Networks

This thesis covered various network topologies to investigate the effects of varying size of the demand set, the mean delay bound and the network topology. Yet only one network (the avionics network) is an actual representative of critical systems. Furthermore, the latency distributions of the remaining networks may not be representative of real delay bounds found in critical systems. Consequently, despite efforts to parametrise both delay bounds and network topologies, it can be argued that the networks and traffic investigated are not representative of those found in safety-critical systems.

However, to the best of the author's knowledge, no further demand sets and corresponding network topologies for safety-critical systems are publicly available. This is most likely because most critical systems are proprietary and hence no corresponding data has been published. The only exception in this context is the demand set used for the avionics network, where at least the distributions and ranges have been published.

6.2 Data Plane

Beyond those general considerations, some additional ones specifically apply to the data plane.

6.2.1 Delay Model

In this thesis, the queueing delay is modelled as a sum of all possible flows over a specific link. Hence, this model assumes that all resilient paths will be used simultaneously. However, in practice, at most two configurations will be served concurrently. Thus, it guarantees that delay bounds will be violated while avoiding having to consider all possible path combinations. Due to the large number of independent demands, considering all combinations would result in a significant number of constraints. This makes the optimisation problem more complex, which may in turn increase computational time.

Furthermore, the optimisations performed in this thesis simplified the delay model by not including transmission, processing, and propagation delay. For the avionics network topology the effect of this simplification on the results is negligible, as the dimension of the network is small. Notwithstanding, this may not be the case for physically larger, safety-critical networks such as the smart grid or the backbone network of an Internet service provider (ISP).

Notwithstanding, accurately modelling network delays is a vast, separate research topic (network calculus) and complementary to the problem addressed in this thesis.

6.2.2 Capacity

The method described in this thesis assumes that the traffic transported is known beforehand. Consequently, it is well-suited for safety-critical systems that do not experience rapid and unexpected variations. Furthermore, it is also suited for those that have to undergo stringent certification procedures, which also require detailed design documents to be submitted such as in aviation.

However, there may also be critical networks where the assumption of a deterministic traffic quantity does not hold. For example, in a smart power grid, the power distribution of renewable energy sources may fluctuate depending on weather. In addition, the demand is also subject to fluctuations based on the time of day or seasons.

Notwithstanding, even in those systems, the margin of error is generally know and accordingly, the optimisation could be performed for several different scenarios (e.g. summer vs. winter). In addition, continuous optimisation may be performed or best-effort heuristics be employed additionally to guarantee best possible service if the current network state differs significantly from worst-case scenarios are not suitable.

Either way, the method presented herein may serve as a basis, even if additional work or adaptations may be needed for some critical systems.

6.2.3 Resilience

The algorithm used in this thesis obtains the maximum number of node- and link-disjoint paths available. This satisfies a key requirement for safety: that no single fault may result in the failure of a safety-critical system/component.

However, it could be argued that it would be more suitable for critical systems to have a dedicated, pre-determined configuration template for every possible failure. This would significantly reduce the capacity requirements and also remove some elements dedicated to deal with unexpected events.

Yet, especially with respect to security, such an approach could be problematic. To begin with, it is very hard to consider all possible issues before deployment. In addition, as previous experience has shown, it is entirely conceivable that potential issues are forgotten or neglected because they are not considered to have a significant impact if they occur.

Furthermore, this approach would also impact safety. While creating individual templates for single failures is an option, it would be increasingly hard to create templates for successive failures. This would require a plethora of different configurations that consider all possible combinations (as detailed in table 4.16), and would consequently also require significant memory.

Beyond that, if the templates are sufficiently different to deal with more than one failure at a time, a reconfiguration would affect all flows and not just those that have been interrupted by the failure. This would require a redesign of the reconfiguration mechanism, as it needs to be ensured that no ongoing transmissions of critical traffic are being interrupted.

In addition, this thesis can address successive, individual failures of network components and respond accordingly. However, this means that only one reconfiguration can be performed at any given time. As long-term failure statistics for SDN hardware are unknown, it may be possible that another failure occurs while the previous reconfiguration is still in progress due to the delay of the consensus. However, different values for k can be assumed if the current requirement that no *single* cause of failure results in catastrophic events proves to be insufficient.

Another issue related to resilience is the fact that the formulation of constraints is thus that all resilient paths could be used concurrently without violations. Consequently, one could argue to simply use all paths concurrently rather than performing a complex reconfiguration that needs additional equipment and processes while providing the same nominative resilience. However, such an approach would be problematic on two counts. Firstly, using the entire TCAM memory would significantly increase energy consumption and can consequently result in insufficient heat dissipation. Secondly, it is also an issue with respect to security, e.g. a denial of service attack on a switch could not be isolated.

6.2.4 Inefficient Resource Usage

Using the optimisation algorithms presented in this thesis to obtain configuration templates ensures that no link and no switch will operate beyond their capacity limit. Yet the resource usage is very unevenly distributed across the network. As shown in chapter 4, some links may not be used at all while others operate near the capacity limit.

This is partially because the resilience constraint is formulated considering that a failure in link e_{ij} also affects the link e_{ji}. While this assumption is incorrect if the link is flooded, it is valid for any physical problem with the cable, e.g. due to chafing. As the latter would consist a single cause of failure (that has to be prevented in critical system) the more conservative formulation has been selected.

6.3 Control Plane

One of the fundamental aspects of this thesis is that no distinction is made between safety and security incidents, while previous work only considers one or the other. Fundamentally, it is a decision between cause and effect. When focussing on the cause, the options with respect to responses are limited. Moreover, it is generally only possible to observe an anomaly and not the cause thereof. Even if some attacks have profiles that cannot be observed for unintentional failures, the focus of this thesis is on *mitigating* the effects of the failure.

Consequently, for most incidents, the distinction is irrelevant. Whether a switch failed because it overheated or because it has been infected with malicious software, a network reconfiguration is needed to restore resilience. However, more complex attacks – especially when several components fail simultaneously – may require specific treatment. Notwithstanding, it is not generally the case that an independent consideration is necessary, especially in cases where only the result and not the cause of a detected anomaly can be identified.

6.4 Management Plane

6.4.1 Safety

The mechanism presented in this thesis allows to select a response that is proportionate to the currently observed network state. Moreover, if a reconfiguration is deemed necessary, *configuration templates* ensure that the requirements of safety-critical traffic are not violated. This allows to tolerate more failures and security incidents than current systems. Notwithstanding, the system is not predictable in the sense that has traditionally been valued in the certification processes of critical systems, which requires fully deterministic systems.

With respect to safety, the usage of SDN could be considered to be problematic, as it has not been designed for safety-critical systems. Consequently, the mean time between failures might be lower than for currently used systems. Some of the more serious effects can be mitigated through the use of the mechanism presented in this thesis, still the question remains why such a system should be used in a safety-critical environment. Yet currently certified systems do not provide any way to adapt to a changing environment, be it due to safety or to security incidents.

As already mentioned (cf. chapter 1), adaptions to running safety-critical systems are complex, because the switches have to be physically accessed to adapt to new operational environments. Especially with respect to cyber-physical safety-critical systems, the continuous evolution and interconnection with the internet increases the exposure of critical systems to cyber risks. Given how long they are in operation and how quickly threats evolve, it would be exceedingly complex to keep non-adaptable systems working reliably. Consequently, with an emerging exposure to security threats, current, unadaptable system may in fact be less reliable overall. However, numeric analyses of this relationship are necessary in the scope of certification, even if they are beyond the scope of this thesis.

Hence, it is necessary that a balance between adaptability and reliability is found in safety critical systems. Previously, certification authorities for some critical systems only had minimal requirements with respect to security [53]. However, various international and national bodies are considering the introduction of more stringent requirements. Yet effective cyber defence requires adaptability lest it becomes predictable to the attacker. Consequently a paradigm change needs to occur towards adaptability, even if individual components may be less reliable.

6.4.2 Security

With respect to security, the approach presented in this thesis enables to protect a safety-critical system from the *effects* of an attack. Notwithstanding, the attack(er) itself may still be present in the system, which could result in further damage. However, more complex defences on the management plane that address the attack(er) itself would require resources that may not be readily available. Even in traditional IT systems, there is a constant evolution of attacks, which may result in defences lagging behind. Due to the lengthy certification process, this effect is aggravated in safety-critical systems. Those are more likely to use legacy devices with low computational powers and may operate with critical vulnerabilities for a longer time since any patch would require to demonstrate non-interference to existing systems.

Beyond that, mobile systems such as aircraft or vehicles are also limited in the resources they can provide to such demanding computations.

Furthermore, the mechanism developed in this thesis has been designed as a means to ensure safety. Thus, security is not desirable for its own sake, but to allow continued safe operation. Consequently, it is of greater importance that the effect of a vulnerability can be mitigated than actively combatting the attack. This is also why potential reaction sets have been grouped by their impact on safety. If a reaction would cause a reduction in safety margins, it must not be performed, even if this would allow an attack to continue.

Chapter 7

Summary and Conclusions

Networks currently in use for safety-critical systems such as the electricity grid or on-board vehicular networks rely on network switches to implement policies and routing strategies. In addition, the most critical paths (e.g. to provide braking) are usually backed up by further, hard wired paths that are not part of the network. However, such networks are limited with respect to complexity, lack of scalability and implementations can vary between different vendors. As critical systems may be in operation for decades, these limits severely impact on the ability of such systems to adapt to changing operational conditions as well as future, potentially more bandwidth-hungry demands.

Yet new strategies and technologies such as SDN would permit more flexible network management by separating the traffic handling from network management. Consequently, switches are only responsible for traffic forwarding and the traffic management is handled by one or more separate entities, the *controller*. As the controller is aware of the current network state, this arrangement can be used to adaptively react to changes in the network, caused by either faulty components or by security incidents.

Hence, more failures may be safely tolerated without the need to enter a degraded mode or postpone the need for a maintenance action. This would allow a continued, safe operation until a fault or vulnerability can be repaired, e.g. allowing an aircraft to complete its journey rather than forcing it to perform an emergency landing. In addition, if this mechanism results in a sufficiently reliable network, hard wired links may safely be removed thus reducing weight – highly relevant for mobile platforms such as automobiles or aircraft – and complexity.

Consequently, the main research question that has been addressed in this thesis is *How can safety-critical networks react dynamically and safely to failures?* This research question has been further subdivided into obtaining candidate responses (configuration templates) and how to apply these responses safely (network configurations) to address detected anomalies.

This chapter provides a summary of the main findings of this thesis. It is divided into three sections that correspond to the main parts of this thesis: section 7.1 summarises the basic system model, while section 7.2 and section 7.3 summarise the generation of configuration templates and running configurations, respectively.

7.1 System Model

To this end, the overall system model has been introduced in chapter 3. Here, the network of a safety-critical system has been modelled as the plant of a standard control system, with failures and attacks acting as disturbances. The SDN controller as well as IDS and IVHM act as sensors in this context. The system controller consists of components that address safety and security issues such as reconfigurations of the firewall as well as the SDN controller that can address data plane failures, irrespective of the cause.

Beyond that, it is clearly undesirable that a minor change in network state results in a complex reconfiguration of the entire network. Consequently, the measure of *appropriateness* has been defined. This ensures that the severity of the response is proportional to that of the detected anomaly. To this end, subsets of reactions are created according to the impact, and matched to a corresponding system state. Hence, only less severe reactions are available to the controller if the system is safe and secure, while all reactions are available if the system is not.

However, dynamic and best-effort strategies are not acceptable for critical systems, as they would result in unpredictable behaviour especially with respect to availability and delay bounds. Consequently, this thesis proposes the use of *configuration templates* that are calculated prior to deployment to guarantee that all demands of safety-critical traffic are satisfied and demonstrated to certification authorities.

7.2 Generating Configuration Templates

This general model is further detailed in section 3.3, where the network model is provided. Beyond that, the routing problem for a safety-critical system is given. To this end, both general networking constraints and constraints specific to the use of SDN in critical system were discussed.

In addition, five primary objective functions (*minimum cost*, *minimum latency*, *minimum number of forwarding table entries*, *maximum resilience* as well as a variant on *minimum link utilisation*) have been detailed.

Beyond that, two heuristic algorithms (EDF and constrained capacity) have been introduced and discussed. Both are based on Dijkstra's shortest path algorithm, but either order the demands with respect to the delay bound and route accordingly or remove links from the network topology if they are either exceeding capacity.

Subsequently, in chapter 4 these algorithms and functions have been used to generate network configurations. The resulting configurations were then compared with respect to their impact on several performance characteristics (bandwidth of links and switches, calculation time, cost, delay, forwarding table entries, memory usage during calculation and path length). To allow fair comparison between the optimisation algorithms when multiple solutions exists, secondary optimisations for all network performance characteristics have been performed when they were not the primary objective function.

It has been found that while heuristics can result in good performance with respect to resource usage and also provide results quickly, they violate timing constraints and are thus unsuitable for use in critical systems. Here, it could be observed that the templates generated with heuristics are not suitable for safety-critical systems as the routings thus obtained would violate timing constraints. While the EDF algorithm produced the smallest number of such violations, they are still unacceptable for safety-critical systems.

However, as significant resources are required to solve the formulated optimisation problems, it is not feasible to perform them on demand when a failure or security incident has been detected. Consequently, candidate routings together with other possible reactions to anomalies need to be calculated prior to deployment.

In contrast, the templates generated using optimisation all satisfied the given requirements. Notwithstanding, significant differences with respect to network performance could be observed. It was found that *minimum forwarding table entries* and *minimum queueing delay* require significantly longer calculation times and only provide modest improvement of their respective optimisation goal compared to *minimum cost.* While *minimum link usage* is calculated much more quickly, other performance characteristics are more similar to *maximum resilience* than the other minimisations, even though the latter provides more paths.

Based on these results, a mechanism has been described in section 4.6 to ensure that safety-critical systems have suitable candidate responses at their disposal when anomalies are detected. As the *maximum resilience* also provides a maximum failure tolerance, the templates thus generated were used to recover from failures in a network emulated using `mininet`. To this end, the two shortest paths were selected as an initial configuration, and additional paths were used to obtain an additional backup path in case a component of the primary paths fail. This permits to provide additional resilience and hence a safe continuation of network service for the critical system as only one path is reconfigured at any time.

7.3 Generating Running Configurations

The final part of this thesis describes how the resulting configuration templates can be used to react to detected network anomalies.

To this end, methods are discussed in chapter 5 to ensure that configuration templates are safely translated into running configurations to mitigate the effect of detected anomalies. To accomplish this, the safety-critical avionics network has been replicated in the `mininet` network emulator.

In this experimental framework, several control plane properties were studied in section 5.1. Here, the success of the mitigation with the amount of flows that can be rerouted as well as the necessary amount of time have been investigated. Furthermore, the effect on mitigation when several failures occur successively has been examined. Beyond that, the amount of control traffic that needs to be exchanged has been studied in order to establish the suitability of an in-band control network. Subsequently, suitable strategies to delete forwarding rules prior to the reconfiguration and the impact thereof have been detailed. It has been found that link failures have a less serious impact as fewer flows are affected. In contrast, as switch failures also affect the connection to end-devices, it is much more likely that a demand cannot be rerouted. Moreover, the number of previous failures only increases the number of unrecoverable flows up to a certain point (10 failures for the avionics network). Afterwards, as the number of end-devices still attached to the network decreases, the number of affected flows drops.

Similar considerations for the management plane were made in section 5.2. The effect of using configuration templates has been studied especially with respect to safety and security considerations. While the use of SDN can improve data plane security by being able to isolate affected components, due care needs to be exerted to ensure that malicious controllers do not cause unauthorised and unnecessary reconfigurations. As one centralised controller would be a single point of failure, several controllers are connected to form a *logically centralised controller*. Consequently, the focus here has been on coordination and control between these different controllers as a means to ensure safety and security. Notwithstanding, it should be noted that for safety-critical systems, security is one of several means to ensure safety.

In addition, safety demands that a minor anomaly such as a slight increase in queueing delay does not result in a major reconfiguration. Consequently, a model has been introduced that matches the potential impact of the reaction to the potential benefit thereof. This ensures that events can be handled appropriately.

7.4 Possible Future Work

With respect to possible further work, the following points are discussed in more detail: certification, delay model, hardware, and multiple failures.

7.4.1 Certification

One key next step in order to use the system presented in this thesis is certification.

If the configuration templates are obtained using linear optimisation, it is guaranteed that the safety constraints are fulfilled if a solution exists. However, more work is needed to demonstrate to certification authorities that the reconfiguration is safe with respect to the control and management plane.

Currently, certification authorities such as the European Aviation Safety Agency (EASA) and FAA consider determinism to be a key factor in deeming a new system to be safe.

As neither the consensus algorithm nor the implementation of the rerouting follow this approach, future work could provide measures to demonstrate that non-deterministic systems can still be safe.

While the measure of appropriateness guarantees that there are no disproportionate responses, it needs to be demonstrated notwithstanding that the system is sufficiently robust not to trigger reconfigurations when they are unnecessary. However, as all configuration templates are guaranteed to be safe, non-determinism in this process should not cause significant issues.

In addition, for certification, it needs to be demonstrated that the reconfiguration process itself is safe. In this context, special care needs to be taken to ensure that no forwarding rules for critical traffic are deleted while traffic is still en-route. Moreover, if the control traffic is exchanged in-band and could interfere with safety-critical traffic, it needs to be demonstrated that control traffic does not interrupt critical transmissions.

7.4.2 Delay Model

As mentioned previously, this thesis uses a very conservative delay model. Consequently, the work presented in this thesis could be extended by using less pessimistic calculations for delay. One way to do this could be to use the methods from network calculus where probabilities are used to obtain more realistic delays.

As the usage of individual paths has been mostly constrained by the delay bound of more critical flows, more flows could benefit from shorter paths, making routing more efficient.

In addition, a less pessimistic delay model could also result in more available paths, as more flows could be handled without violating constraints.

Furthermore, the results of chapter 5 could be used to analyse which paths are served concurrently. This analysis could then be used to obtain the most likely combinations of demands that concurrently use the same resources. Hence, a more realistic queueing model could be obtained without having to iterate over all possible demand combinations.

7.4.3 Hardware

Beyond that, future work could include measuring the reliability of specific hardware in order to investigate the reliability. While the system presented in this thesis can address a higher failure rate of network components than current systems, obtaining concrete data for failure probability could be relevant to assure certification authorities that the system is safe.

In addition, if the failure probability of the network is sufficiently low, additional hardwired redundancy could be removed to reduce weight and complexity.

7.4.4 Multiple, Simultaneous Failures

This thesis only considers one fault to occur at any given time. However, especially security incidents may result in more than one failure, e.g. if switches of one manufacturer share a firmware vulnerability.

While hard- and software diversification should protect the system from a critical failure in such a case, future work could build upon this thesis by considering multiple failures at any one time. This would be an especially worthwhile undertaking in the context that multiple failures would also have a more serious impact on the network than a single failure.

Chapter A

OpenFlow Control Plane Messages

Table A.1: Summary of Controller-to-Switch Messages

Request	**Reply**	**Frequency**
Features	Identity and general features	Generally upon establishing the OF channel
Configuration (Set/ Query)	Config. params	Unspecified
Modify-State (flow tables, switch port properties)	-	As necessary
Read-State	Configs, statistics, abilities (gen. multipart)	As necessary
Packet-out of specific port	-	As necessary
Barrier (relation between messages)	-	Upon changes/terminated operations
Role-Request of a controller	None	As necessary
Asynchronous Message Filtering (if multiple controllers)	None	Generally upon establishing the OF channel

Table A.2: Summary of Asynchronous Messages

Request	**Reply**	**Frequency**
Transfer packet control	Possible: packet-out	As necessary
Flow-Removed	-	Upon `OFPFF_SEND_FLOW_REM`
Port-Status	-	Upon changes of port status (e.g. Port down, link down)
Error	-	When problems occur

Table A.3: Summary of Synchronous Messages

Request	**Reply**	**Frequency**
Hello	Hello	Upon establishing connection
Echo	Echo	To verify connection/measuring latency and bandwidth
Experimenter	-	placeholder for additional functionalities in future versions

Chapter B

Source Files

Listing B.1: Network Initialisation

```
import time
import thread
import random
import argparse
import numpy as np
import logging
import copy

from mininet.topo import Topo, MultiGraph
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.node import RemoteController, Ryu,OVSKernelSwitch, UserSwitch, Host
from mininet.util import dumpNodeConnections
from mininet.clean import cleanup
from mininet.log import lg as mininet_log
from mininet.link import Link, TCLink

import json
import requests as rq

import networkx as nx
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

from networkx.readwrite import json_graph
import websocket
from itertools import combinations
import bottle
from multiprocessing import Process

from functools import partial

from comlib import *


""" To change location of configuration files, search for "TODO"
"""

""" Global settings:
"""
# Set global log level
log = logging.getLogger("mininet")
# Url for Ryu controller REST interface
```

```
rc_rest = 'http://localhost:8080'


def bottlerun(seed=None, **kwargs):
    """ Wrapper to initialize the random number generator in the bottle process
    """
    if seed != None:
        random.seed(seed)
        np.random.seed(seed)
    bottle.run(**kwargs)


def bottlewrap(*dargs, **dkwargs):
    """ Decorator to nicely include functions which return boolean in bottle
    """
    success_proba = 1
    if 'success_proba' in dkwargs:
        success_proba = dkwargs['success_proba']

    def decorate(f):
        @bottle.route(*dargs, **dkwargs)
        def bottled(*args, **kwargs):
            if success_proba < 1 and random.random() > success_proba:
                bottle.abort(501, 'random␣failure␣of␣' + f.__name__)

            ret = f(*args, **kwargs)
            if type(ret) == bool:
                if ret:
                    return 'OK'
                else:
                    bottle.abort(500, f.__name__ + '␣returned␣False')
            return ret

        return f

    return decorate
"""Helper function to determine whether mininet is up and running remotely
"""
@bottle.route('/ping')
def bping(mn=None):
    if mn is None:
        global gmn
        mn = gmn
    if mn is None:
        bottle.abort(500, "Mininet␣not␣initialized␣yet")
    if not mn.built:
        bottle.abort(500, "Mininet␣not␣built␣yet")
    if hasattr(mn, "all_booted") and not mn.all_booted:
        bottle.abort(500, "Mininet␣not␣fully␣booted␣yet")
    return "pong"

# ----------------------------------------------------------------------------
"""Helper function to obtain network topology remotely
"""
@bottlewrap('/get_topology')
def get_topology(mn=None):
    if mn is None:
        global gmn
        mn = gmn
    G = build_graph(mn, as_str=False)
    return json_graph.node_link_data(G)

@bottlewrap('/get_mac_<name>')
def get_host_mac(name):
    global gmn
    return gmn.host(name).MAC()

""" Helper function to clear all flow entries from a given switch
"""

@bottlewrap('/clear_all_flow_entries/<dpid>')
def clear_all_flow_entries(dpid):
    """Remove all flow entries from a given switch
    """
    r = rq.delete(rc_rest + ('/stats/flowentry/clear/%d' % int(dpid)))
    return r.status_code == 200


""" Remove flow entries from killed switch and stop it:
"""
@bottlewrap('/kill_switch_<number>')
def kill_switch(number):
    failed_switch=number.replace("(", "").replace(")","")
    req_status=clear_all_flow_entries(int(gmn.switch('s'+str(failed_switch)).dpid))
    gmn.switch('s'+number).stop()
    # gmn.switch('s'+number).delSwitch()
    get_topology(mn=gmn)
    return req_status

@bottlewrap('/node/<name>/stop')
def node_stop(name):
    global gmn
    return gmn.nameToNode[name].stop(deleteIntfs=False)

@bottlewrap('/write_mininet_topo')
def write_mininet_topo():
    global gmn
    to={'nodes': gmn.topo.g.node, 'edges': gmn.topo.g.edge}

    with open('/tmp/mininet_topo.json', 'w') as f:
        json.dump(to, f, indent=2)
    with open('/tmp/mininet_ips', 'w') as f:
        for name, node in gmn.nameToNode.iteritems():
            f.write(name + "\t" + str(node.IP()) + "\n")


@bottlewrap('/node/<name>/start', success_proba=0.9999)
def node_start(name):
    global gmn
    return gmn.nameToNode[name].start(gmn.controllers)


@bottlewrap('/link/<node1>/<node2>/down')
def link_down(node1, node2):
    global gmn
    gmn.configLinkStatus(node1, node2, "down")


@bottlewrap('/link/<node1>/<node2>/up', sucess_proba=0.9999)
def link_up(node1, node2):
    global gmn
    gmn.configLinkStatus(node1, node2, "up")


@bottlewrap('/switch/<dpid>/start', success_proba=0.9999)
def switch_start(dpid):
    global gmn
    gmn.dpidToNode[dpid].start(gmn.controllers)
    return install_static_routes()


@bottle.route('/network_status')
def network_status(mn=None):
    if mn is None:
        global gmn
        mn = gmn
    h = {}

    G = build_graph(mn)
    connectivity = []
    for i, j in combinations(mn.switches, 2):
        if i not in G or j not in G:
            continue
        connectivity.append(nx.local_node_connectivity(G, i, j))
    if len(connectivity) > 0:
        h['connectivity_switches_mean'] = np.mean(connectivity)
        h['connectivity_switches_min'] = np.min(connectivity)
    else:
        h['connectivity_switches_mean'] = 0
        h['connectivity_switches_min'] = 0

    connectivity = []
    for i, j in combinations(mn.hosts, 2):
        connectivity.append(nx.local_node_connectivity(G, i, j))
    if len(connectivity) > 0:
        h['connectivity_hosts_mean'] = np.mean(connectivity)
        h['connectivity_hosts_min'] = np.min(connectivity)
    else:
        h['connectivity_hosts_mean'] = 0
        h['connectivity_hosts_min'] = 0

    num_active_switches = 0
    for s in mn.switches:
        r = rq.get(rc_rest + ('/stats/desc/%d' % int(s.dpid)))
        if r.status_code != 200:
            continue
        num_active_switches += 1
    h['availability_switches'] = num_active_switches / float(len(mn.switches))

    return h

# ----------------------------------------------------------------------------------------
# Build network graph
@bottlewrap('/build_graph')
def build_graph(mn=None, save_graph_png=True, as_str=False, plot=True):
    title='Initial␣network'
    if mn==None:
        global gmn
        mn = gmn
        title= 'Current␣topology'
    """Build the graph of the network
    """
    if as_str:
        transf = lambda x : x # transf= strl # possible alternative: str(len(x))
    else:
        transf = lambda x : x
    G = nx.Graph()
    nodes = set()
    for s in mn.switches:
        r = rq.get(rc_rest + ('/stats/desc/%d' % int(s.dpid)))
        if r.status_code != 200:
            continue
        G.add_node(transf(s), type="switch")
        nodes.add(s)

    for l in mn.links:
        if not l.intf1.isUp():
            continue
        if l.intf1.node not in nodes:
            continue
        if l.intf2.node not in nodes:
            continue
        G.add_edge(transf(l.intf1.node), transf(l.intf2.node), interfaces={transf(l.intf1.node
            ): transf(l.intf1), transf(l.intf2.node): transf(l.intf2)})


        # Save a picture of the network

    # nx.draw(G, hosts, nodecolor='b')
    if plot:
        """
        hosts=[]
        switches=[]
        for h in mn.hosts:
            G.add_node(transf(h), type="host")
            nodes.add(h)
        for node in G.nodes():
            if G.node[node]['type']=='host':
                hosts.append(G.node[node])
            elif G.node[node]['type']=='switch':
                switches.append(G.node[node])
        """

        if save_graph_png:
            plt.figure()
            nx.draw_networkx(G) # networkx draw()
            plt.savefig("nx.png")
        else:
            plt.show()
    return G


def afdx_topology(network, configuration=None):
    "Add␣switches"
    def get_switch(nr):
        if nr==1:
            switch=switch1
        if nr==2:
            switch=switch2
        if nr==3:
            switch=switch3
        if nr==4:
            switch=switch4
        if nr==5:
            switch=switch5
        if nr==6:
            switch=switch6
        if nr==7:
            switch=switch7
        if nr==8:
            switch=switch8
        return switch

    switch1=network.addSwitch('s1')
    switch2=network.addSwitch('s2')
    switch3=network.addSwitch('s3')
    switch4=network.addSwitch('s4')
    switch5=network.addSwitch('s5')
    switch6=network.addSwitch('s6')
    switch7=network.addSwitch('s7')
    switch8=network.addSwitch('s8')


    "Add␣links␣between␣switches"
    # Eq. for port nr: input/output(1/2)*100+ source_switch*10+ target_switch*1
    t=network.addLink(switch1, switch8, port1=118, port2=218)
    network.addLink(switch8, switch1, port1=181, port2=281)
    network.addLink(switch1, switch2, port1=112, port2=212)
    network.addLink(switch2, switch1, port1=121, port2=221)
    network.addLink(switch1, switch3, port1=113, port2=213)
    network.addLink(switch3, switch1, port1=131, port2=231)

    network.addLink(switch2, switch8, port1=128, port2=228)
    network.addLink(switch8, switch2, port1=182, port2=282)
    network.addLink(switch2, switch4, port1=124, port2=224)
    network.addLink(switch4, switch2, port1=142, port2=242)

    network.addLink(switch3, switch8, port1=138, port2=238)
    network.addLink(switch8, switch3, port1=183, port2=283)
    network.addLink(switch3, switch4, port1=134, port2=234)
    network.addLink(switch4, switch3, port1=143, port2=243)
    network.addLink(switch3, switch5, port1=135, port2=235)
    network.addLink(switch5, switch3, port1=153, port2=253)
    network.addLink(switch3, switch7, port1=137, port2=237)
    network.addLink(switch7, switch3, port1=173, port2=273)

    network.addLink(switch4, switch8, port1=148, port2=248)
    network.addLink(switch8, switch4, port1=184, port2=284)
    network.addLink(switch4, switch6, port1=146, port2=246)
    network.addLink(switch6, switch4, port1=164, port2=264)
    network.addLink(switch4, switch7, port1=147, port2=247)
    network.addLink(switch7, switch4, port1=174, port2=274)
    network.addLink(switch5, switch6, port1=156, port2=256)
    network.addLink(switch6, switch5, port1=165, port2=265)
    network.addLink(switch5, switch7, port1=157, port2=257)
    network.addLink(switch7, switch5, port1=175, port2=275)

    network.addLink(switch6, switch7, port1=167, port2=267)
    network.addLink(switch7, switch6, port1=176, port2=276)

    return network

def create_hosts_and_links(gmn, configuration, config_type):

    for demand in configuration[config_type]:
        # print(demand)
        vlan_base = get_vlan_base(config_type)
        source, destination, NULL = parse_input(configuration, config_type, demand)
        h_s = gmn.addHost(get_host_name(config_type, demand, s_t='s'), cls=VLANHost, vlan=
            vlan_base+int(demand))
        h_t = gmn.addHost(get_host_name(config_type, demand, s_t='t'), cls=VLANHost, vlan=
            vlan_base+int(demand))
        # Eq. for port number: 1/2*100+vlanbase+demand
        gmn.addLink(h_s, gmn.switch('s'+str(source)),
                    port1=100+vlan_base+int(demand),
                    port2=200+vlan_base+int(demand))

        gmn.addLink(gmn.switch('s'+str(destination)), h_t,
                    port1=100+vlan_base+int(demand), port2=200+vlan_base+int(demand))
        with open('/tmp/mininet_mac', 'a') as f:
            #print(str(get_host_name(config_type, demand, s_t='s')) + "\t" + str(h_s.MAC()) +
                '\n')
            #print(str(get_host_name(config_type, demand, s_t='t')) + "\t" + str(h_t.MAC()) +
                '\n')
            f.write(str(get_host_name(config_type, demand, s_t='s'))+"\t"+str(h_s.MAC())+'\n')
            f.write(str(get_host_name(config_type, demand, s_t='t'))+"\t"+str(h_t.MAC())+'\n')

    return gmn

def main(args):
    global gmn
    global configuration

    # Cleanup mininet before starting
    cleanup()

    #TODO change paths here
    configuration=read_config(filename="/home/tancs/initial_configuration.csv",
                  default_filename="initial_configuration.csv")
    # END TODO

    # Remote controller
    # > ryu-manager --observe-links ryu.app.gui_topology.gui_topology ryu.app.rest_conf_switch
    #of_ctrlr = RemoteController('c0', ip='127.0.0.1', port=6633)
    of_ctrlr = Ryu('ryu', '--observe-links', 'ryu.app.gui_topology.gui_topology', 'ryu.app.
        rest_conf_switch')

    gmn = Mininet(controller=of_ctrlr,topo=afdx_topology(Topo()), autoStaticArp=True, link=
        TCLink, switch=OVSKernelSwitch)#,switch=OVSKernelSwitch)#added, link=TCLink,switch=
        OVSKernelSwitch link=.., switch, removed topology and autoStaticArp=true


    gmn.all_booted = False



    # Start mininet
    gmn.start()

    # Small pause in order for mininet to bootup everything
    gmn.waitConnected()
    gmn=create_hosts_and_links(gmn, configuration, 'standard')
    gmn=create_hosts_and_links(gmn, configuration, 'backup')
    to={'nodes': gmn.topo.g.node, 'edges': gmn.topo.g.edge}
    with open('/tmp/mininet_topo.json', 'w') as f:
        json.dump(to, f, indent=2)
    with open('/tmp/mininet_ips', 'w') as f:
        for name, node in gmn.nameToNode.iteritems():
            f.write(name + "\t" + str(node.IP()) + "\n")

    gmn.dpidToNode = {}
    with open('/tmp/mininet_dpid', 'w') as f:
        for sw in gmn.switches:
            gmn.dpidToNode[sw.dpid] = sw
            gmn.dpidToNode[int(sw.dpid)] = sw
            f.write(sw.name + "\t" + sw.dpid + "\n")
    gmn.all_booted = True
    # Export topology for attacker and recovery

    # Starts the webserver for remote controlling mininet


    # Install routes for standard and backup path

    forwarding_rule_counter= install_forwarding_table_entries(configuration,config_type="
         standard", reconf=False)
    print(bcolors.HEADER + "Installed "+str(forwarding_rule_counter)+ " forwarding rules on 
         switches for standard path" + bcolors.ENDC)
    forwarding_rule_counter= install_forwarding_table_entries(configuration,config_type="
         backup", reconf=False)
    print(bcolors.HEADER + "Installed "+str(forwarding_rule_counter)+ " forwarding rules on 
         switches for backup path" + bcolors.ENDC)

    bottle.debug()
    h = Process(target=bottlerun,
                kwargs=dict(seed=random.getrandbits(32), host='0.0.0.0', port=8888))  # ,
                    quiet=not args.bottle_debug))
    h.start()

    print(bcolors.OKGREEN + "Mininet up. \nInitialising forwarding tables." + bcolors.ENDC)
    # Custom command line interface with added commands
    class CLIext(CLI):
        def do_kill_switch(self, number):
            "Call install_static_routes()"
            kill_switch(number)
    # Starts the command line interface

    get_topology(mn=gmn)
    CLIext(gmn)

    gmn.stop()
    h.terminate()

# ----------------------------------------------------------------------------

if __name__ == "__main__":
    with open('/tmp/mininet_mac', 'w') as f:
        f.write("")
    parser = argparse.ArgumentParser(description='Mininet wrapper')
    parser.add_argument('--seed', help='Seed for random.seed', type=int, default=0)
    parser.add_argument('--verbose', help='Verbose logs', action='store_true')
    parser.add_argument('--bottle-debug', help='Adds logging to bottle', action='store_true')
    args = parser.parse_args()
    # little trick to ensure that list of failed components is empty at the start of the
         program
    with open('/home/ubuntu-lab/Documents/decade_demo/home/mininet/demo_files/failed.txt', 'w'
         ) as f:
        f.write('')
    random.seed(args.seed)
    np.random.seed(args.seed)

    logging.basicConfig()
    if args.verbose:
        log.setLevel(logging.DEBUG)
        #mininet_log.setLogLevel("debug")
        mininet_log.setLogLevel("info")
    else:
        log.setLevel(logging.INFO)
        logging.getLogger("urllib3").setLevel(logging.ERROR)

    main(args)
```

Listing B.2: Adaptive Controller

```
#!/usr/bin/python

import time
import random
import argparse
import requests as rq
import logging
import numpy as np
from comlib import *
from ryu.controller import ofp_event
from ryu.controller.handler import set_ev_cls

from multiprocessing import Process

import websocket as websk  # https://github.com/liris/websocket-client
import time

log = logging.getLogger("controller")

# ------------------------------------------------------------------------------

# Mininet REST API base url
mn_url = 'http://127.0.0.1:8888'
# Url for Ryu controller REST interface
rc_rest = 'http://localhost:8080'
ryu_ws_url = 'ws://localhost:8080/v1.0/topology/ws'

switch_leave_recall_delay = 2
link_delete_recall_delay = 2

""" Reroute when switch failure has been detected, using either new configuration
    that has been added by consensus of the distributed ledger or the default file
"""
def reroute_switch_failure(switch_id):
    failed_switch=str(int(switch_id))
    recover_from_failure(failed_switch)
    return

def reroute_link_failure(name):
    recover_from_failure(name.replace('s', ""))
    return

def get_network_status():
    r = rq.get(mn_url + "/network_status")
    return json.loads(r.text)

"""def get_topology_differences(expected_topo, current_topo):
    failed={'s':[], 'l':[]}
    # compare failure-free network with current network configuration
    print(expected_topo.switches())
    print(current_topo.switches())
    for s in expected_topo.switches():
        if s not in set(current_topo.switches()):
            failed['s']+=[s]
    for l in expected_topo.g.edge:
        if l not in set(current_topo.g.edge):
           failed['l']+=l
    print failed
    return failed
"""

def recover_from_failure(failed):

    #TODO change paths here
    configuration=read_config(filename="/home/tancs/initial_configuration.csv",
                              default_filename="initial_configuration.csv")
    # END TODO
    log.info(bcolors.FAIL+'Failure␣detected!'+bcolors.ENDC)
    # rq.get(mn_url+'/build_graph')
    if len(failed)<5:
        failure_type = 'switch␣'
    else:
        failure_type = 'link␣'
    log.info(bcolors.HEADER+'Recovering␣from␣failure␣of␣'+failure_type +failed+"␣..."+bcolors.
        ENDC)
    counter_deleted=0
    try:
        all_failed=get_identifiers('failed')
    except IOError:
        all_failed={}
    all_failed[failed]={}
    # create dictionary of demands that need to be rerouted
    demands_2_reroute={'standard':{}, 'backup':{}}
    failed_demands={'standard':{}, 'backup':{}}
    unrecoverable={'standard':0, 'backup':0}
    forwarding_rule_counter={'standard':0, 'backup':0}
    # get previous failures from file
    prev_failed=[]
    with open('/home/ubuntu-lab/Documents/decade_demo/home/mininet/demo_files/results_2.csv',
         'r') as f:
        for line in f:
            prev_failed.append(line.rstrip("\n"))
    # identify which routes are affected by the failure
    for config_type in ['standard', 'backup']:
        for demands in configuration[config_type]:
            # check whether current demand is affected by the failure
            # NB: this will cease to work if the switch names can are non unique e.g. 11 and 1
            if configuration[config_type][demands].find(failed)!=-1:
                counter_deleted+=1
                failed_demands[config_type][demands]=configuration[config_type][demands]
                # iterate over alternative routes in the configuration file
                for alternatives in range(len(configuration['remainder'][demands])):
                    """ check whether alternatives are affected by switch failure,
                        either because of another (unrelated) failure, or because
                        a switch connecting to a host is affected
                    """

                    altern = [False for i in range(len(all_failed))]
                    fc=0
                    # check allternative routes for failed components
                    if demands not in demands_2_reroute[config_type]:
                        for failed in all_failed:
                            if configuration['remainder'][demands][alternatives].find(failed)
                                ==-1:
                                for pf in prev_failed:
                                    if configuration['remainder'][demands][alternatives].find(
                                        pf)==-1:
                                        altern[fc] = True
                            fc+=1
                        if altern == [True for i in range(len(all_failed))]:
                            demands_2_reroute[config_type][demands]=configuration['remainder'
                                ][demands][alternatives]
                    """

                    if configuration['remainder'][demands][alternatives].find(failed)==-1:
                        demands_2_reroute[config_type][demands]=configuration['remainder'][
                            demands][alternatives]
                    """

                if demands not in demands_2_reroute[config_type]:
                     unrecoverable[config_type]+=1
    if counter_deleted>0:
        log.info(bcolors.HEADER+str(counter_deleted)+'␣demand(s)␣need␣to␣be␣rerouted'+bcolors.
            ENDC)
        for configs in demands_2_reroute:
            if demands_2_reroute[configs]!={}:
                forwarding_rule_counter[configs]= install_forwarding_table_entries(
                    demands_2_reroute,config_type=configs, reconf=False)
                log.info(bcolors.OKGREEN+"Installed␣"+str(forwarding_rule_counter[configs])+ "
                    ␣forwarding␣rules␣to␣recover␣"+configs+ "␣routes.␣\n"+bcolors.ENDC)
        for ura in unrecoverable:
            if unrecoverable[ura]>0:
                log.warn(bcolors.FAIL+str(unrecoverable[ura])+'␣demand(s)␣could␣not␣be␣
                    rerouted␣for␣the␣'+ ura+ '␣routes.'+bcolors.ENDC)
            else:
                log.info(bcolors.OKGREEN+'␣All␣demands␣could␣be␣rerouted␣for␣the␣'+ura+'␣
                    routes.'+bcolors.ENDC)

    else:
        log.info(bcolors.OKGREEN+'No␣demand(s)␣need␣to␣be␣rerouted!␣Failure␣does␣not␣affect␣
            running␣configuration.␣'+bcolors.ENDC)
    with open('/home/ubuntu-lab/Documents/decade_demo/home/mininet/demo_files/results_2.csv',
        'a') as f:
        f.write(failed.replace(",", "-")+","+str(counter_deleted)+","+ str(
            forwarding_rule_counter['standard'])+","+ str(forwarding_rule_counter['backup'])+
            ","+str(unrecoverable['standard'])+","+ str(unrecoverable['backup'])+"\n")
    # write failed components to file
    with open('/home/ubuntu-lab/Documents/decade_demo/home/mininet/demo_files/failed.txt', 'a'
        ) as f:
        f.write(failed+'\n')
    return unrecoverable


last_sensor_val = [0]
action_args = {}

last_switch_leave_event = {}
last_link_event = {}

def opf_msg(ws, msg):
    global last_sensor_val

    # Acknowledges the message
    ws.send('{"id":␣1,␣"jsonrpc":␣"2.0",␣"result":␣""}')
    obj = json.loads(msg)
    log.debug(obj)
    # log.info("openflow: " + obj["method"])

    ts = time.time()

    if obj["method"] == "event_switch_leave":
        # Record the last event from this switch
        dpid = obj["params"][0]["dpid"]
        last_switch_leave_event[dpid] = {"ts": ts, "params": obj["params"][0]}

        last_sensor_val = [1]
        reroute_switch_failure(dpid)
        Process(target=rq.get(mn_url+'/build_graph'))
        return

    if obj["method"] == "event_link_delete":
        # We first need to determine if this link event is linked to a switch which has failed
        dpid1 = obj["params"][0]["dst"]["dpid"]
        if dpid1 in last_switch_leave_event:
            deltat = ts - last_switch_leave_event[dpid1]["ts"]
            if deltat < switch_leave_recall_delay:
                return

        dpid2 = obj["params"][0]["src"]["dpid"]
        if dpid2 in last_switch_leave_event:
            deltat = ts - last_switch_leave_event[dpid2]["ts"]
            if deltat < switch_leave_recall_delay:
                return

        # We determine if we have already treated the link event or not, since for each
            Ethernet link failure, we get an event for both directions
        key = (obj["params"][0]["dst"]["name"], obj["params"][0]["src"]["name"])
        if key in last_link_event:
            deltat = ts - last_link_event[key]["ts"]
            if deltat < link_delete_recall_delay:
                return

        last_link_event[(key[1], key[0])] = {"ts": ts}
        node1 = obj["params"][0]["src"]["name"]
        node1 = node1[:node1.find("-eth")]
        node2 = obj["params"][0]["dst"]["name"]
        node2 = node2[:node2.find("-eth")]
        last_sensor_val = [2]
        reroute_link_failure("(" + str(node1) + ", " + str(node2) + ")")
        Process(target=rq.get(mn_url+'/build_graph'))
        return

def opf_onopen(ws):
    log.info('Connection with OpenFlow controller established')
    ws.opf_ok = True

def opf_onclose(ws):
    if hasattr(ws, "opf_ok") and ws.opf_ok:
        log.warn(bcolors.WARNING+'Connection with OpenFlow controller closed'+bcolors.ENDC)

def main(retry_delay=2):
    check_mininet_status(mn_url=mn_url, log=log)

    log.info('Trying to connect to OpenFlow controller')
    while 1:
        ws = websk.WebSocketApp(ryu_ws_url, on_message=opf_msg, on_open=opf_onopen, on_close=
            opf_onclose)
        try:
            ws.run_forever()
        except KeyboardInterrupt:
            break
        time.sleep(retry_delay)

# ---------------------------------------------------------------------------

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Mininet wrapper')
    parser.add_argument('--seed', help='Seed for random number generator', type=int, default
        =0)
    parser.add_argument('--verbose', help='More verbose logs', action='store_true')
    args = parser.parse_args()

    random.seed(args.seed)
    np.random.seed(args.seed)

    logging.basicConfig()
    if args.verbose:
        log.setLevel(logging.DEBUG)
    else:
        log.setLevel(logging.INFO)
        logging.getLogger("urllib3").setLevel(logging.WARNING)

    main()
```

Listing B.3: Attacking Script

```
#!/usr/bin/python

import time
import random
import argparse
import requests as rq
import logging
import numpy as np
from comlib import *
from multiprocessing import Process

log = logging.getLogger("attacker")

# ---------------------------------------------------------------------------

# Mininet REST API base url
mn_url = 'http://127.0.0.1:8888'

# ---------------------------------------------------------------------------

"""
def remote_mininet(path, **kwargs):
    r = rq.get(mn_url + path, **kwargs)
    if r.status_code == 200:
        return True
    log.warn("Remote mininet: " + path + "\n" + r.text)
    return False
"""

def stop_node(name):
    return remote_mininet('/node/%s/stop' % name)


def link_down(node1, node2):
    return remote_mininet('/link/%s/%s/down' % (node1, node2))

# ---------------------------------------------------------------------------


def nodestop_chaos_loop(seed, nodes, failrate):
    if seed != None:
        # Note: We only need to seed the random number generator if we start this function in
             a new process
        random.seed(seed)

    while 1:
        deltat = 1. / random.expovariate(failrate)
        log.info(bcolors.HEADER+'next␣attack␣on␣switches␣in␣'+ str(int(deltat)) +'␣s' +bcolors
             .ENDC)
        time.sleep(deltat)
        node = random.choice(nodes)
        log.info(bcolors.FAIL+'␣targeting␣switch␣' + node+bcolors.ENDC)
        stop_node(node)
        time.sleep(2)


def linkdown_chaos_loop(seed, links, failrate):
    if seed != None:
        # Note: We only need to seed the random number generator if we start this function in
             a new process
        random.seed(seed)

    while 1:
        deltat = 1. / random.expovariate(failrate)
        log.info(bcolors.HEADER+'next␣attack␣on␣links␣in␣'+ str(int(deltat)) +'␣s' +bcolors.
            ENDC)
        time.sleep(deltat)
        link = random.choice(links)
        output=str(link).replace('u',"").replace('s', '').replace('\'', '')
        log.info(bcolors.FAIL+'␣targeting␣link' + str(output)+bcolors.ENDC)
        link_down(*link)
        time.sleep(2)

# ---------------------------------------------------------------------------


def main(args):

    check_mininet_status(mn_url=mn_url, log=log)

    topo, ipsl = read_mininet_topo()

    if args.sw_failrate > 0:
        Process(target=nodestop_chaos_loop, args=(random.getrandbits(8), topo.switches(), args
             .sw_failrate)).start()

    if args.link_failrate > 0:
        Process(target=linkdown_chaos_loop, args=(random.getrandbits(9), topo.links(), args.
            link_failrate)).start()

# ---------------------------------------------------------------------------

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Mininet attacker')
    parser.add_argument('--seed', help='Seed for random.seed', type=int, default=0)
    parser.add_argument('--verbose', help='Verbose logs', action='store_true')
    parser.add_argument('--sw-failrate', help='Failure rate of switches', type=float, default
        =20)
    parser.add_argument('--host-failrate', help='Failure rate of hosts', type=float, default
        =0)
    parser.add_argument('--link-failrate', help='Failure rate of links', type=float, default
        =20)
    args = parser.parse_args()

    random.seed(args.seed)
    np.random.seed(args.seed)

    logging.basicConfig()
    if args.verbose:
        log.setLevel(logging.DEBUG)
    else:
        log.setLevel(logging.DEBUG)
        logging.getLogger("urllib3").setLevel(logging.WARNING)

    main(args)
```

Chapter C

Additional Results of Comparison of Algorithms

This chapter provides for the network topologies that have not been included in the main part of this thesis because limited new information could be gained from them. For completeness' sake, they are included herein.

C.1 Heuristics

As the varying delay bound is not taken into account by Dijkstra's algorithm or the limited capacity, it is not included in the results detailed in table C.1 to table C.7. For the mean number of violations, no difference could be observed from varying the delay bound. Note that the *nobel-germany* network has been abbreviated to n-g in some tables.

C.2 Optimisation

As shown in the tables (table C.8 to table C.15) below, there is no difference for most investigated parameters between the different delay bounds. In these cases, only the results minimum and maximum delay bounds are listed. The comparison where the (albeit small) impact of a varying delay bound could be observed is for the number of resilient paths.

C.3 Comparison Between Optimisation and Heuristics

In figure C.1 and figure C.2, the results for the optimisation for *minimum cost* have been obtained using the GLPK solver. Hence, they are denoted as such in the figures. In addition, they contain results from Python's `shortest paths` algorithm. As they are identical for all the cases investigated, they have been omitted for the remainder of the results.

Table C.1: Calculation Time Heuristics

Network	Heuristic	Mean Delay	Min.	Median	Mean	Max.
atlanta	dijkstra	-	0.153	0.162	0.166	0.221
atlanta	EDF	630	0.144	0.158	0.16	0.186
atlanta	EDF	840	0.14	0.149	0.155	0.178
atlanta	EDF	1050	0.138	0.143	0.144	0.16
atlanta	EDF	1260	0.136	0.144	0.149	0.192
atlanta	EDF	1470	0.137	0.141	0.145	0.16
atlanta	EDF	1680	0.135	0.142	0.143	0.156
atlanta	EDF	1890	0.137	0.143	0.143	0.148
atlanta	EDF	2100	0.14	0.141	0.144	0.159
atlanta	Lim. capa.	-	0.153	0.162	0.166	0.221
dfn	dijkstra	-	0.086	0.1	0.103	0.131
dfn	EDF	90	104.152	106.236	108.376	116.568
dfn	EDF	180	103.644	104.777	105.901	109.526
dfn	EDF	270	99.723	101.031	101.123	103.343
dfn	EDF	360	99.343	100.896	101.015	103.976
dfn	EDF	450	97.943	100.557	100.508	103.73
dfn	EDF	540	98.455	100.662	101.506	108.979
dfn	EDF	630	98.28	102.766	102.721	105.965
dfn	EDF	720	97.733	102.81	102.038	105.098
dfn	EDF	810	97.797	101.193	102.518	114.052
dfn	EDF	900	99.761	100.893	100.851	102.926
dfn	Lim. capa.	-	0.086	0.1	0.103	0.131
nobel-germany	dijkstra	-	0.098	0.106	0.121	0.316
nobel-germany	EDF	484	0.211	0.263	0.29	0.549
nobel-germany	EDF	605	0.212	0.234	0.237	0.265
nobel-germany	EDF	726	0.176	0.216	0.219	0.272
nobel-germany	EDF	847	0.173	0.178	0.181	0.214
nobel-germany	EDF	968	0.179	0.183	0.183	0.192
nobel-germany	EDF	1089	0.175	0.182	0.182	0.191
nobel-germany	EDF	1210	0.174	0.179	0.183	0.214
nobel-germany	Lim. capa.	-	0.098	0.106	0.121	0.316

Table C.2: Path Length Heuristics

Network	**Heuristic**	**Mean Delay**	**Min.**	**Median**	**Mean**	**Max.**
atlanta	dijkstra	-	1	3	3.348	7
atlanta	EDF	-	1	3	3.037	8
atlanta	Lim. capa.	-	1	3	3.34	7
dfn	dijkstra	-	1	1.5	1.5	2
dfn	EDF	-	1	2	1.509	3
dfn	Lim. capa.	-	1	1.5	1.5	2
nobel-germany	dijkstra	-	1	3	3.393	8
nobel-germany	EDF	-	1	3	3.081	8
nobel-germany	Lim. capa.	-	1	3	3.393	8

Table C.3: Queueing Delay Heuristics

Network	**Heuristic**	**Mean Delay**	**Min.**	**Median**	**Mean**	**Max.**
atlanta	Dijkstra	-	30	291	294.819	709
atlanta	EDF	630	34	205	209.419	597
atlanta	EDF	2100	34	205	209.419	597
atlanta	Lim. capa.	-	30	286.5	292.883	704
dfn	Dijkstra	-	2	16	14.433	34
dfn	EDF	90	4	10	9.211	21
dfn	EDF	900	4	10	9.211	21
dfn	Lim. capa.	-	2	16	14.433	34
nobel-germany	Dijkstra	-	13	153	163.066	464
nobel-germany	EDF	484	13	124	127.902	334
nobel-germany	EDF	1210	13	124	127.902	334
nobel-germany	Lim. capa.	-	13	153	163.066	464

Table C.4: Link Usage Heuristics

Network	Heuristic	Mean Delay	Min.	Median	Mean	Max.
dfn	Dijkstra	-	$1.413 \cdot 10^3$	$18.44 \cdot 10^3$	$36.56 \cdot 10^3$	$199.2 \cdot 10^3$
dfn	EDF	90	$2.5 \cdot 10^3$	$23.26 \cdot 10^3$	$36.76 \cdot 10^3$	$191.2 \cdot 10^3$
dfn	EDF	900	$2.5 \cdot 10^3$	$23.26 \cdot 10^3$	$36.76 \cdot 10^3$	$191.2 \cdot 10^3$
dfn	Lim. capa.	-	$1.413 \cdot 10^3$	$18.44 \cdot 10^3$	$36.56 \cdot 10^3$	$199.2 \cdot 10^3$
n-g	Dijkstra	-	44	142	189.4	500
n-g	EDF	484	44	144	176.8	510
n-g	EDF	1210	44	144	176.8	510
n-g	Lim. capa.	-	44	142	189.4	500
atlanta	Dijkstra	-	$8.488 \cdot 10^3$	$34.99 \cdot 10^3$	$45.48 \cdot 10^3$	$114.6 \cdot 10^3$
atlanta	EDF	630	$10.9 \cdot 10^3$	$32.75 \cdot 10^3$	$41.58 \cdot 10^3$	$108.1 \cdot 10^3$
atlanta	EDF	2100	$10.9 \cdot 10^3$	$32.75 \cdot 10^3$	$41.58 \cdot 10^3$	$108.1 \cdot 10^3$
atlanta	Lim. capa.	-	$8.488 \cdot 10^3$	$35.86 \cdot 10^3$	$45.15 \cdot 10^3$	$111.4 \cdot 10^3$

Table C.5: Switch Usage Heuristics

Network	Heuristic	Mean Delay	Min.	Median	Mean	Max.
dfn	Dijkstra	-	$140.5 \cdot 10^3$	$250.4 \cdot 10^3$	$274.2 \cdot 10^3$	$814.1 \cdot 10^3$
dfn	EDF	90	$131.8 \cdot 10^3$	$275 \cdot 10^3$	$330.8 \cdot 10^3$	$868.5 \cdot 10^3$
dfn	EDF	900	$131.8 \cdot 10^3$	$275 \cdot 10^3$	$330.8 \cdot 10^3$	$868.5 \cdot 10^3$
dfn	Lim. capa.	-	$140.5 \cdot 10^3$	$250.4 \cdot 10^3$	$274.2 \cdot 10^3$	$814.1 \cdot 10^3$
n-g	Dijkstra	-	126	224	301.5	740
n-g	EDF	484	132	314	421	$1.02 \cdot 10^3$
n-g	EDF	1210	132	314	421	$1.02 \cdot 10^3$
n-g	Lim. capa.	-	126	224	301.5	740
atlanta	Dijkstra	-	$16.29 \cdot 10^3$	$55.6 \cdot 10^3$	$69.89 \cdot 10^3$	$166 \cdot 10^3$
atlanta	EDF	-	$23.16 \cdot 10^3$	$81.86 \cdot 10^3$	$97.24 \cdot 10^3$	$212.2 \cdot 10^3$
atlanta	Lim. capa.	-	$16.29 \cdot 10^3$	$53.42 \cdot 10^3$	$69.67 \cdot 10^3$	$164 \cdot 10^3$

Table C.6: Forwarding Table Entries Heuristics

Network	Heuristic	Mean Delay	Min.	Median	Mean	Max.
dfn	Dijkstra	-	36	44.5	45	62
dfn	EDF	90	50	54	54.328	62
dfn	EDF	900	50	54	54.328	62
dfn	Lim. capa.	-	36	44.5	45	62
nobel-germany	Dijkstra	-	10	56	62.529	128
nobel-germany	EDF	484	28	64	87.708	204
nobel-germany	EDF	1210	28	64	87.708	204
nobel-germany	Lim. capa.	-	10	56	62.529	128
atlanta	Dijkstra	-	58	122	121.733	199
atlanta	EDF	630	74	174	170.047	290
atlanta	EDF	2100	74	174	170.047	290
atlanta	Lim. capa.	-	58	121	121.533	200

Table C.7: Constraint Violations Heuristics

Network	Heuristic	Mean Delay	Min.	Median	Mean	Max.
nobel-germany	Dijkstra	484	1	1	1.4	2
nobel-germany	Dijkstra	1210	1	1	1.4	2
nobel-germany	EDF	484	0	0	0	0
nobel-germany	Lim. capa.	484	1	1	1.4	2
nobel-germany	Lim. capa.	1210	1	1	1.4	2
atlanta	Dijkstra	630	9	9	9.256	13
atlanta	Dijkstra	2100	9	9	9.256	13
atlanta	EDF	630	8	9	9.35	11
atlanta	EDF	2100	8	9	9.35	11
atlanta	Lim. capa.	630	9	9	9.167	12
atlanta	Lim. capa.	2100	9	9	9.167	12

Table C.8: Calculation Times Optimisation

Network	**Function**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	Min. link. util	90	16.42	16.42	16.42	16.42
dfn	Min. link. util	900	16.42	16.42	16.42	16.42
dfn	Max. res.	90	86.13	172.3	594.9	$5.112{\cdot}10^3$
dfn	Max. res.	900	86.13	172.3	594.9	$5.112{\cdot}10^3$
dfn	Min. cost	90	19.63	63.26	59.15	276.5
dfn	Min. cost	900	19.63	63.26	59.15	276.5
dfn	Min. fwd. rules	90	19.64	31.16	49.29	204.8
dfn	Min. fwd. rules	900	19.64	31.16	49.29	204.8
dfn	Min. delay	90	5.2	138.3	$4.156{\cdot}10^3$	$34.77{\cdot}10^3$
dfn	Min. delay	900	5.2	138.3	$4.156{\cdot}10^3$	$34.77{\cdot}10^3$
n-g	Min. link. util	484	47.85	47.85	47.85	47.85
n-g	Min. link. util	1210	47.85	47.85	47.85	47.85
n-g	Max. res.	484	21.28	21.82	$4.988{\cdot}10^3$	$35.21{\cdot}10^3$
n-g	Max. res.	1210	21.28	21.82	$4.988{\cdot}10^3$	$35.21{\cdot}10^3$
n-g	Min. cost	484	32.57	60.83	66.89	136.4
n-g	Min. cost	1210	32.57	60.83	66.89	136.4
n-g	Min. fwd. rules	484	31.98	42.55	45.39	118.2
n-g	Min. fwd. rules	1210	31.98	42.55	45.39	118.2
n-g	Min. delay	484	2.2	104.9	$5.479{\cdot}10^3$	$89.86{\cdot}10^3$
n-g	Min. delay	1210	2.2	104.9	$5.479{\cdot}10^3$	$89.86{\cdot}10^3$
atlanta	Min. link. util	630	526.5	526.5	526.5	526.5
atlanta	Min. link. util	2100	526.5	526.5	526.5	526.5
atlanta	Max. res.	630	43.23	113.3	567.8	$56.76{\cdot}10^3$
atlanta	Max. res.	2100	43.23	113.3	567.8	$56.76{\cdot}10^3$
atlanta	Min. cost	630	126	$2.068{\cdot}10^3$	$5.97{\cdot}10^3$	$154{\cdot}10^3$
atlanta	Min. cost	2100	126	$2.068{\cdot}10^3$	$5.97{\cdot}10^3$	$154{\cdot}10^3$
atlanta	Min. fwd. rules	630	121	242.5	$1.631{\cdot}10^3$	$155.5{\cdot}10^3$
atlanta	Min. fwd. rules	2100	121	242.5	$1.631{\cdot}10^3$	$155.5{\cdot}10^3$
atlanta	Min. delay	630	3.9	105.2	$4.539{\cdot}10^3$	$154.7{\cdot}10^3$
atlanta	Min. delay	2100	3.9	105.2	$4.539{\cdot}10^3$	$154.7{\cdot}10^3$

Table C.9: Cost Optimisation

Network	**Function**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	Min. link. util	90	$48.21{\cdot}10^9$	$48.21{\cdot}10^9$	$48.21{\cdot}10^9$	$48.21{\cdot}10^9$
dfn	Min. link. util	900	$48.21{\cdot}10^9$	$48.21{\cdot}10^9$	$48.21{\cdot}10^9$	$48.21{\cdot}10^9$
dfn	Max. res.	90	$161.2{\cdot}10^9$	$172.8{\cdot}10^9$	$172.3{\cdot}10^9$	$172.8{\cdot}10^9$
dfn	Max. res.	900	$161.2{\cdot}10^9$	$172.8{\cdot}10^9$	$172.3{\cdot}10^9$	$172.8{\cdot}10^9$
dfn	Min. cost	90	$30.2{\cdot}10^9$	$30.2{\cdot}10^9$	$30.2{\cdot}10^9$	$30.2{\cdot}10^9$
dfn	Min. cost	900	$30.2{\cdot}10^9$	$30.2{\cdot}10^9$	$30.2{\cdot}10^9$	$30.2{\cdot}10^9$
dfn	Min. fwd. rules	90	$30.23{\cdot}10^9$	$30.23{\cdot}10^9$	$30.23{\cdot}10^9$	$30.23{\cdot}10^9$
dfn	Min. fwd. rules	900	$30.23{\cdot}10^9$	$30.23{\cdot}10^9$	$30.23{\cdot}10^9$	$30.23{\cdot}10^9$
dfn	Min. delay	90	$30.22{\cdot}10^9$	$30.22{\cdot}10^9$	$30.22{\cdot}10^9$	$30.22{\cdot}10^9$
dfn	Min. delay	900	$30.22{\cdot}10^9$	$30.22{\cdot}10^9$	$30.22{\cdot}10^9$	$30.22{\cdot}10^9$
n-g	Min. link. util	484	$354.7{\cdot}10^6$	$354.7{\cdot}10^6$	$354.7{\cdot}10^6$	$354.7{\cdot}10^6$
n-g	Min. link. util	1210	$354.7{\cdot}10^6$	$354.7{\cdot}10^6$	$354.7{\cdot}10^6$	$354.7{\cdot}10^6$
n-g	Max. res.	484	$441.2{\cdot}10^6$	$441.2{\cdot}10^6$	$441.2{\cdot}10^6$	$441.2{\cdot}10^6$
n-g	Max. res.	1210	$441.2{\cdot}10^6$	$441.2{\cdot}10^6$	$441.2{\cdot}10^6$	$441.2{\cdot}10^6$
n-g	Min. cost	484	$302.7{\cdot}10^6$	$302.7{\cdot}10^6$	$302.8{\cdot}10^6$	$304.3{\cdot}10^6$
n-g	Min. cost	1210	$302.7{\cdot}10^6$	$302.7{\cdot}10^6$	$302.8{\cdot}10^6$	$304.3{\cdot}10^6$
n-g	Min. fwd. rules	484	$304.7{\cdot}10^6$	$304.7{\cdot}10^6$	$304.7{\cdot}10^6$	$304.7{\cdot}10^6$
n-g	Min. fwd. rules	1210	$304.7{\cdot}10^6$	$304.7{\cdot}10^6$	$304.7{\cdot}10^6$	$304.7{\cdot}10^6$
n-g	Min. delay	484	$304{\cdot}10^6$	$304{\cdot}10^6$	$304{\cdot}10^6$	$304{\cdot}10^6$
n-g	Min. delay	1210	$304{\cdot}10^6$	$304{\cdot}10^6$	$304{\cdot}10^6$	$304{\cdot}10^6$
atlanta	Min. link. util	630	$1.524{\cdot}10^{12}$	$1.524{\cdot}10^{12}$	$1.524{\cdot}10^{12}$	$1.524{\cdot}10^{12}$
atlanta	Min. link. util	2100	$1.524{\cdot}10^{12}$	$1.524{\cdot}10^{12}$	$1.524{\cdot}10^{12}$	$1.524{\cdot}10^{12}$
atlanta	Max. res.	630	$1.512{\cdot}10^{12}$	$1.636{\cdot}10^{12}$	$1.628{\cdot}10^{12}$	$1.73{\cdot}10^{12}$
atlanta	Max. res.	2100	$1.512{\cdot}10^{12}$	$1.636{\cdot}10^{12}$	$1.628{\cdot}10^{12}$	$1.73{\cdot}10^{12}$
atlanta	Min. cost	630	$1.006{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.009{\cdot}10^{12}$
atlanta	Min. cost	2100	$1.006{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.009{\cdot}10^{12}$
atlanta	Min. fwd. rules	630	$1.014{\cdot}10^{12}$	$1.015{\cdot}10^{12}$	$1.015{\cdot}10^{12}$	$1.017{\cdot}10^{12}$
atlanta	Min. fwd. rules	2100	$1.014{\cdot}10^{12}$	$1.015{\cdot}10^{12}$	$1.015{\cdot}10^{12}$	$1.017{\cdot}10^{12}$
atlanta	Min. delay	630	$978.6{\cdot}10^9$	$978.6{\cdot}10^9$	$978.6{\cdot}10^9$	$978.6{\cdot}10^9$
atlanta	Min. delay	2100	$978.6{\cdot}10^9$	$978.6{\cdot}10^9$	$978.6{\cdot}10^9$	$978.6{\cdot}10^9$

Table C.10: Path Length Optimisation

Network	**Function**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	Min. link. util	90	1	2	2.211	6
dfn	Min. link. util	900	1	2	2.211	6
dfn	Max. res.	90	1	2	1.889	7
dfn	Max. res.	900	1	2	1.889	7
dfn	Min. cost	90	1	1.5	1.5	2
dfn	Min. cost	900	1	1.5	1.5	2
dfn	Min. fwd. rules	90	1	1.5	1.5	2
dfn	Min. fwd. rules	900	1	1.5	1.5	2
dfn	Min. delay	90	1	2	1.507	3
dfn	Min. delay	900	1	2	1.507	3
n-g	Min. link. util	484	1	4	3.909	12
n-g	Min. link. util	1210	1	4	3.909	12
n-g	Max. res.	484	1	4	4.728	15
n-g	Max. res.	1210	1	4	4.728	15
n-g	Min. cost	484	1	3	3.36	8
n-g	Min. cost	1210	1	3	3.36	8
n-g	Min. fwd. rules	484	1	3	3.36	8
n-g	Min. fwd. rules	1210	1	3	3.36	8
n-g	Min. delay	484	1	3	3.414	10
n-g	Min. delay	1210	1	3	3.414	10
atlanta	Min. link. util	630	1	4	3.983	10
atlanta	Min. link. util	2100	1	4	3.983	10
atlanta	Max. res.	630	1	4	4.324	16
atlanta	Max. res.	2100	1	4	4.324	16
atlanta	Min. cost	630	1	3	3.345	9
atlanta	Min. cost	2100	1	3	3.345	9
atlanta	Min. fwd. rules	630	1	3	3.332	7
atlanta	Min. fwd. rules	2100	1	3	3.332	7
atlanta	Min. delay	630	1	4	3.836	13
atlanta	Min. delay	2100	1	4	3.836	13

Table C.11: Queueing Delay Optimisation

Network	**Function**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	Min. link. util	90	1	28	34.144	97
dfn	Min. link. util	900	1	28	34.144	97
dfn	Max. res.	90	19	83	87.272	310
dfn	Max. res.	900	19	83	87.272	310
dfn	Min. cost	90	1	8	7.345	17
dfn	Min. cost	900	1	8	7.345	17
dfn	Min. fwd. rules	90	1	6	6.422	23
dfn	Min. fwd. rules	900	1	6	6.422	23
dfn	Min. delay	90	1	4	4.933	8
dfn	Min. delay	900	1	4	4.933	8
n-g	Min. link. util	484	4	96.5	104.917	272
n-g	Min. link. util	1210	4	96.5	104.917	272
n-g	Max. res.	484	10	138	154.519	492
n-g	Max. res.	1210	10	138	154.519	492
n-g	Min. cost	484	3	78.5	89.021	256
n-g	Min. cost	1210	3	78.5	89.021	256
n-g	Min. fwd. rules	484	3	81	89.789	254
n-g	Min. fwd. rules	1210	3	81	89.789	254
n-g	Min. delay	484	2	100	107.488	230
n-g	Min. delay	1210	2	100	107.488	230
atlanta	Min. link. util	630	16	163	177.112	427
atlanta	Min. link. util	2100	16	163	177.112	427
atlanta	Max. res.	630	17	143	164.333	758
atlanta	Max. res.	2100	17	143	164.333	758
atlanta	Min. cost	630	6	133	134.342	332
atlanta	Min. cost	2100	6	133	134.342	332
atlanta	Min. fwd. rules	630	9	129	130.706	301
atlanta	Min. fwd. rules	2100	9	129	130.706	301
atlanta	Min. delay	630	12	126	128.164	296
atlanta	Min. delay	2100	12	126	128.164	296

Table C.12: Switch Utilisation Optimisation

Network	**Function**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	Min. link. util	90	$210.1{\cdot}10^3$	$514.2{\cdot}10^3$	$521.2{\cdot}10^3$	$846{\cdot}10^3$
dfn	Min. link. util	900	$210.1{\cdot}10^3$	$514.2{\cdot}10^3$	$521.2{\cdot}10^3$	$846{\cdot}10^3$
dfn	Max. res.	90	$1.32{\cdot}10^6$	$1.622{\cdot}10^6$	$1.861{\cdot}10^6$	$3.892{\cdot}10^6$
dfn	Max. res.	900	$1.32{\cdot}10^6$	$1.622{\cdot}10^6$	$1.861{\cdot}10^6$	$3.892{\cdot}10^6$
dfn	Min. cost	90	$112.8{\cdot}10^3$	$257.4{\cdot}10^3$	$329{\cdot}10^3$	$902.3{\cdot}10^3$
dfn	Min. cost	900	$112.8{\cdot}10^3$	$257.4{\cdot}10^3$	$329{\cdot}10^3$	$902.3{\cdot}10^3$
dfn	Min. fwd. rules	90	$117.9{\cdot}10^3$	$268.9{\cdot}10^3$	$329{\cdot}10^3$	$840.6{\cdot}10^3$
dfn	Min. fwd. rules	900	$117.9{\cdot}10^3$	$268.9{\cdot}10^3$	$329{\cdot}10^3$	$840.6{\cdot}10^3$
dfn	Min. delay	90	$206.8{\cdot}10^3$	$429.4{\cdot}10^3$	$453.3{\cdot}10^3$	$872.8{\cdot}10^3$
dfn	Min. delay	900	$206.8{\cdot}10^3$	$429.4{\cdot}10^3$	$453.3{\cdot}10^3$	$872.8{\cdot}10^3$
n-g	Min. link. util	484	244	372	522.4	$1.096{\cdot}10^3$
n-g	Min. link. util	1210	244	372	522.4	$1.096{\cdot}10^3$
n-g	Max. res.	484	540	832	843.1	$1.44{\cdot}10^3$
n-g	Max. res.	1210	540	832	843.1	$1.44{\cdot}10^3$
n-g	Min. cost	484	172	300	445.2	$1{\cdot}10^3$
n-g	Min. cost	1210	172	300	445.2	$1{\cdot}10^3$
n-g	Min. fwd. rules	484	156	272	445.2	$1.004{\cdot}10^3$
n-g	Min. fwd. rules	1210	156	272	445.2	$1.004{\cdot}10^3$
n-g	Min. delay	484	332	452	627.3	$1.328{\cdot}10^3$
n-g	Min. delay	1210	332	452	627.3	$1.328{\cdot}10^3$
atlanta	Min. link. util	630	$27.38{\cdot}10^3$	$147.5{\cdot}10^3$	$137.8{\cdot}10^3$	$219.7{\cdot}10^3$
atlanta	Min. link. util	2100	$27.38{\cdot}10^3$	$147.5{\cdot}10^3$	$137.8{\cdot}10^3$	$219.7{\cdot}10^3$
atlanta	Max. res.	630	$43.88{\cdot}10^3$	$180.3{\cdot}10^3$	$177.1{\cdot}10^3$	$310.6{\cdot}10^3$
atlanta	Max. res.	2100	$43.88{\cdot}10^3$	$180.3{\cdot}10^3$	$177.1{\cdot}10^3$	$310.6{\cdot}10^3$
atlanta	Min. cost	630	$16.98{\cdot}10^3$	$84.02{\cdot}10^3$	$103.3{\cdot}10^3$	$213.5{\cdot}10^3$
atlanta	Min. cost	2100	$16.98{\cdot}10^3$	$84.02{\cdot}10^3$	$103.3{\cdot}10^3$	$213.5{\cdot}10^3$
atlanta	Min. fwd. rules	630	$16.98{\cdot}10^3$	$82.31{\cdot}10^3$	$103.3{\cdot}10^3$	$218.9{\cdot}10^3$
atlanta	Min. fwd. rules	2100	$16.98{\cdot}10^3$	$82.31{\cdot}10^3$	$103.3{\cdot}10^3$	$218.9{\cdot}10^3$
atlanta	Min. delay	630	$43.11{\cdot}10^3$	$134.1{\cdot}10^3$	$128.7{\cdot}10^3$	$243.4{\cdot}10^3$
atlanta	Min. delay	2100	$43.11{\cdot}10^3$	$134.1{\cdot}10^3$	$128.7{\cdot}10^3$	$243.4{\cdot}10^3$

Table C.13: Link Utilisation Optimisation

Network	**Function**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	Min. link. util	90	0	$1.746 \cdot 10^3$	$28.96 \cdot 10^3$	$264.7 \cdot 10^3$
dfn	Min. link. util	900	0	$1.746 \cdot 10^3$	$28.96 \cdot 10^3$	$264.7 \cdot 10^3$
dfn	Max. res.	90	$33.43 \cdot 10^3$	$78.54 \cdot 10^3$	$103.4 \cdot 10^3$	$283.2 \cdot 10^3$
dfn	Max. res.	900	$33.43 \cdot 10^3$	$78.54 \cdot 10^3$	$103.4 \cdot 10^3$	$283.2 \cdot 10^3$
dfn	Min. cost	90	289	$3.368 \cdot 10^3$	$18.28 \cdot 10^3$	$135 \cdot 10^3$
dfn	Min. cost	900	289	$3.368 \cdot 10^3$	$18.28 \cdot 10^3$	$135 \cdot 10^3$
dfn	Min. fwd. rules	90	126	$8.331 \cdot 10^3$	$18.28 \cdot 10^3$	$227.3 \cdot 10^3$
dfn	Min. fwd. rules	900	126	$8.331 \cdot 10^3$	$18.28 \cdot 10^3$	$227.3 \cdot 10^3$
dfn	Min. delay	90	$1.3 \cdot 10^3$	$18.46 \cdot 10^3$	$24.81 \cdot 10^3$	$128.6 \cdot 10^3$
dfn	Min. delay	900	$1.3 \cdot 10^3$	$18.46 \cdot 10^3$	$24.81 \cdot 10^3$	$128.6 \cdot 10^3$
n-g	Min. link. util	484	0	82	85.38	210
n-g	Min. link. util	1210	0	82	85.38	210
n-g	Max. res.	1210	32	118	129.8	334
n-g	Min. cost	484	4	65	72.77	222
n-g	Min. cost	1210	4	65	72.77	222
n-g	Min. fwd. rules	484	2	61	72.77	220
n-g	Min. fwd. rules	1210	2	61	72.77	220
n-g	Min. delay	484	0	113	104.2	196
n-g	Min. delay	1210	0	113	104.2	196
atlanta	Min. link. util	630	0	$26.68 \cdot 10^3$	$23.48 \cdot 10^3$	$43.65 \cdot 10^3$
atlanta	Min. link. util	2100	0	$26.68 \cdot 10^3$	$23.48 \cdot 10^3$	$43.65 \cdot 10^3$
atlanta	Max. res.	630	$9.057 \cdot 10^3$	$31.43 \cdot 10^3$	$30.86 \cdot 10^3$	$40 \cdot 10^3$
atlanta	Max. res.	2100	$9.057 \cdot 10^3$	$31.43 \cdot 10^3$	$30.86 \cdot 10^3$	$40 \cdot 10^3$
atlanta	Min. cost	630	$4.134 \cdot 10^3$	$15.65 \cdot 10^3$	$17.68 \cdot 10^3$	$40 \cdot 10^3$
atlanta	Min. cost	2100	$4.134 \cdot 10^3$	$15.65 \cdot 10^3$	$17.68 \cdot 10^3$	$40 \cdot 10^3$
atlanta	Min. fwd. rules	630	$4.134 \cdot 10^3$	$16.27 \cdot 10^3$	$17.6 \cdot 10^3$	$40 \cdot 10^3$
atlanta	Min. fwd. rules	2100	$4.134 \cdot 10^3$	$16.27 \cdot 10^3$	$17.6 \cdot 10^3$	$40 \cdot 10^3$
atlanta	Min. delay	630	$6.074 \cdot 10^3$	$17.77 \cdot 10^3$	$20.64 \cdot 10^3$	$42.47 \cdot 10^3$
atlanta	Min. delay	2100	$6.074 \cdot 10^3$	$17.77 \cdot 10^3$	$20.64 \cdot 10^3$	$42.47 \cdot 10^3$

Table C.14: Forwarding Table Entries Optimisation

Network	**Function**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	Min. link. util	90	26	36.5	39.8	59
dfn	Min. link. util	900	26	36.5	39.8	59
dfn	Max. res.	90	146	153	170.601	277
dfn	Max. res.	900	146	153	170.601	277
dfn	Min. cost	90	18	24	27	40
dfn	Min. cost	900	18	24	27	40
dfn	Min. fwd. rules	90	18	23.5	27	43
dfn	Min. fwd. rules	900	18	23.5	27	43
dfn	Min. delay	90	18	34	32.4	35
dfn	Min. delay	900	18	34	32.4	35
n-g	Min. link. util	484	19	47	55.647	120
n-g	Min. link. util	1210	19	47	55.647	120
n-g	Max. res.	484	23	67	76.444	192
n-g	Max. res.	1210	23	67	76.444	192
n-g	Min. cost	484	10	40	47.824	109
n-g	Min. cost	1210	10	40	47.824	109
n-g	Min. fwd. rules	484	8	38	47.824	107
n-g	Min. fwd. rules	1210	8	38	47.824	107
n-g	Min. delay	1210	30	55	60.941	114
atlanta	Min. link. util	630	43	119	111.533	172
atlanta	Min. link. util	2100	43	119	111.533	172
atlanta	Max. res.	630	34	136	136.061	232
atlanta	Max. res.	2100	34	136	136.061	232
atlanta	Min. cost	630	30	84	94.232	173
atlanta	Min. cost	2100	30	84	94.232	173
atlanta	Min. fwd. rules	630	30	94	93.288	171
atlanta	Min. fwd. rules	2100	30	94	93.288	171
atlanta	Min. delay	630	28	161	127.4	186
atlanta	Min. delay	2100	28	161	127.4	186

Table C.15: Number of Resilient Paths

Network	Function	Mean delay lim.	Min.	Median	Mean	Max.
atlanta	dijkstra	630	1	2	2.322	3
atlanta	dijkstra	840	1	2	2.33	3
atlanta	dijkstra	1050	1	2	2.33	3
atlanta	dijkstra	1260	1	2	2.331	3
atlanta	dijkstra	1470	1	2	2.333	3
atlanta	dijkstra	1680	2	2	2.332	3
atlanta	dijkstra	1890	1	2	2.333	3
atlanta	dijkstra	2100	1	2	2.332	3
dfn	dijkstra	90	9	9	9	9
dfn	dijkstra	180	9	9	9	9
dfn	dijkstra	270	9	9	9	9
dfn	dijkstra	360	5	9	8.984	9
dfn	dijkstra	450	7	9	8.998	9
dfn	dijkstra	540	3	9	8.983	9
dfn	dijkstra	630	9	9	9	9
dfn	dijkstra	720	9	9	9	9
dfn	dijkstra	810	7	9	8.997	9
dfn	dijkstra	900	9	9	9	9
n-g	dijkstra	484	1	2	2.39	4
n-g	dijkstra	605	1	2	2.39	4
n-g	dijkstra	726	1	2	2.388	4
n-g	dijkstra	847	1	2	2.393	4
n-g	dijkstra	968	1	2	2.388	4
n-g	dijkstra	1089	1	2	2.39	4
n-g	dijkstra	1210	1	2	2.392	4

Table C.16: Cost Heuristic vs. Optimisation

Network	**Heuristic**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	EDF	90	$100.4{\cdot}10^{9}$	$105.5{\cdot}10^{9}$	$105.7{\cdot}10^{9}$	$114.2{\cdot}10^{9}$
dfn	EDF	900	$100.4{\cdot}10^{9}$	$105.5{\cdot}10^{9}$	$105.7{\cdot}10^{9}$	$114.2{\cdot}10^{9}$
dfn	Min. cost	90	$30.2{\cdot}10^{9}$	$30.2{\cdot}10^{9}$	$30.2{\cdot}10^{9}$	$30.2{\cdot}10^{9}$
dfn	Min. cost	900	$30.2{\cdot}10^{9}$	$30.2{\cdot}10^{9}$	$30.2{\cdot}10^{9}$	$30.2{\cdot}10^{9}$
nobel-germany	EDF	484	$6.069{\cdot}10^{9}$	$6.279{\cdot}10^{9}$	$6.285{\cdot}10^{9}$	$6.545{\cdot}10^{9}$
nobel-germany	EDF	1210	$6.069{\cdot}10^{9}$	$6.279{\cdot}10^{9}$	$6.285{\cdot}10^{9}$	$6.545{\cdot}10^{9}$
nobel-germany	Min. cost	1210	$302.7{\cdot}10^{6}$	$302.7{\cdot}10^{6}$	$302.7{\cdot}10^{6}$	$304.3{\cdot}10^{6}$
atlanta	EDF	630	$42.89{\cdot}10^{12}$	$44.81{\cdot}10^{12}$	$44.83{\cdot}10^{12}$	$47.05{\cdot}10^{12}$
atlanta	EDF	2100	$42.89{\cdot}10^{12}$	$44.81{\cdot}10^{12}$	$44.83{\cdot}10^{12}$	$47.05{\cdot}10^{12}$
atlanta	Min. cost	630	$1.005{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.009{\cdot}10^{12}$
atlanta	Min. cost	2100	$1.005{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.007{\cdot}10^{12}$	$1.009{\cdot}10^{12}$

Table C.17: Calculation Time Heuristic vs. Optimisation

Network	**Heuristic**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	EDF	90	97.73	101.4	102.7	116.6
dfn	EDF	900	97.73	101.4	102.7	116.6
dfn	Min. cost	90	19.63	63.26	59.15	276.5
dfn	Min. cost	900	19.63	63.26	59.15	276.5
nobel-germany	EDF	484	$173.5{\cdot}10^{-3}$	$184.5{\cdot}10^{-3}$	$210.8{\cdot}10^{-3}$	$548.8{\cdot}10^{-3}$
nobel-germany	EDF	1210	$173.5{\cdot}10^{-3}$	$184.5{\cdot}10^{-3}$	$210.8{\cdot}10^{-3}$	$548.8{\cdot}10^{-3}$
nobel-germany	Min. cost	1210	32.57	60.83	66.89	136.4
atlanta	EDF	630	$135{\cdot}10^{-3}$	$143.3{\cdot}10^{-3}$	$148{\cdot}10^{-3}$	$192.1{\cdot}10^{-3}$
atlanta	EDF	2100	$135{\cdot}10^{-3}$	$143.3{\cdot}10^{-3}$	$148{\cdot}10^{-3}$	$192.1{\cdot}10^{-3}$
atlanta	Min. cost	630	126	$2.068{\cdot}10^{3}$	$5.97{\cdot}10^{3}$	$154{\cdot}10^{3}$
atlanta	Min. cost	2100	126	$2.068{\cdot}10^{3}$	$5.97{\cdot}10^{3}$	$154{\cdot}10^{3}$

Table C.18: Path Length Heuristic vs. Optimisation

Network	**Heuristic**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	EDF	90	1	2	1.509	3
dfn	EDF	900	1	2	1.509	3
dfn	Min. cost	90	0	1.5	1.5	3
dfn	Min. cost	900	0	1.5	1.5	3
nobel-germany	EDF	484	1	3	3.081	8
nobel-germany	EDF	1210	1	3	3.081	8
nobel-germany	Min. cost	1210	1	3	3.36	9
atlanta	EDF	630	1	3	3.037	8
atlanta	EDF	2100	1	3	3.037	8
atlanta	Min. cost	630	1	3	3.349	9
atlanta	Min. cost	2100	1	3	3.349	9

Table C.19: Queuing Delay Heuristic vs. Optimisation

Network	**Heuristic**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	EDF	90	4	10	9.211	21
dfn	EDF	900	4	10	9.211	21
dfn	Min. cost	90	1	8	7.345	17
dfn	Min. cost	900	1	8	7.345	17
nobel-germany	EDF	484	13	124	127.902	334
nobel-germany	EDF	1210	13	124	127.902	334
nobel-germany	Min. cost	1210	3	78	89.018	256
atlanta	EDF	630	34	205	209.419	597
atlanta	EDF	2100	34	205	209.419	597
atlanta	Min. cost	630	5	131	132.72	332
atlanta	Min. cost	2100	5	131	132.72	332

Table C.20: Link Utilisation Heuristic vs. Optimisation

Network	**Heuristic**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	EDF	90	$2.5{\cdot}10^3$	$23.26{\cdot}10^3$	$36.76{\cdot}10^3$	$191.2{\cdot}10^3$
dfn	EDF	900	$2.5{\cdot}10^3$	$23.26{\cdot}10^3$	$36.76{\cdot}10^3$	$191.2{\cdot}10^3$
dfn	Min. cost	90	289	$3.368{\cdot}10^3$	$18.28{\cdot}10^3$	$135{\cdot}10^3$
dfn	Min. cost	900	289	$3.368{\cdot}10^3$	$18.28{\cdot}10^3$	$135{\cdot}10^3$
nobel-germany	EDF	484	44	144	176.8	510
nobel-germany	EDF	1210	44	144	176.8	510
nobel-germany	Min. cost	1210	4	65	72.77	222
atlanta	EDF	630	$10.9{\cdot}10^3$	$32.75{\cdot}10^3$	$41.58{\cdot}10^3$	$108.1{\cdot}10^3$
atlanta	EDF	2100	$10.9{\cdot}10^3$	$32.75{\cdot}10^3$	$41.58{\cdot}10^3$	$108.1{\cdot}10^3$
atlanta	Min. cost	630	$4.134{\cdot}10^3$	$15.28{\cdot}10^3$	$17.65{\cdot}10^3$	$40{\cdot}10^3$
atlanta	Min. cost	2100	$4.134{\cdot}10^3$	$15.28{\cdot}10^3$	$17.65{\cdot}10^3$	$40{\cdot}10^3$

Table C.21: Switch Utilisation Heuristic vs. Optimisation

Network	**Heuristic**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	EDF	90	$131.8{\cdot}10^3$	$275{\cdot}10^3$	$330.8{\cdot}10^3$	$868.5{\cdot}10^3$
dfn	EDF	900	$131.8{\cdot}10^3$	$275{\cdot}10^3$	$330.8{\cdot}10^3$	$868.5{\cdot}10^3$
dfn	Min. cost	90	$112.8{\cdot}10^3$	$257.4{\cdot}10^3$	$329{\cdot}10^3$	$903.7{\cdot}10^3$
dfn	Min. cost	900	$112.8{\cdot}10^3$	$257.4{\cdot}10^3$	$329{\cdot}10^3$	$903.7{\cdot}10^3$
nobel-germany	EDF	484	132	314	421	$1.02{\cdot}10^3$
nobel-germany	EDF	1210	132	314	421	$1.02{\cdot}10^3$
nobel-germany	Min. cost	1210	172	300	445.2	$1{\cdot}10^3$
atlanta	EDF	630	$23.16{\cdot}10^3$	$81.86{\cdot}10^3$	$97.24{\cdot}10^3$	$212.2{\cdot}10^3$
atlanta	EDF	2100	$23.16{\cdot}10^3$	$81.86{\cdot}10^3$	$97.24{\cdot}10^3$	$212.2{\cdot}10^3$
atlanta	Min. cost	630	$16.98{\cdot}10^3$	$82.87{\cdot}10^3$	$103.5{\cdot}10^3$	$216{\cdot}10^3$
atlanta	Min. cost	2100	$16.98{\cdot}10^3$	$82.87{\cdot}10^3$	$103.5{\cdot}10^3$	$216{\cdot}10^3$

Table C.22: Fwd. Tab. Entries Heuristic vs. Optimisation

Network	**Heuristic**	**Mean delay lim.**	**Min.**	**Median**	**Mean**	**Max.**
dfn	EDF	90	50	54	54.328	62
dfn	EDF	900	50	54	54.328	62
dfn	Min. cost	90	18	24	27	40
dfn	Min. cost	900	18	24	27	40
nobel-germany	EDF	484	28	64	87.708	204
nobel-germany	EDF	1210	28	64	87.708	204
nobel-germany	Min. cost	1210	10	40	47.828	109
atlanta	EDF	630	74	174	170.047	290
atlanta	EDF	2100	74	174	170.047	290
atlanta	Min. cost	630	30	83	93.761	176
atlanta	Min. cost	2100	30	83	93.761	176

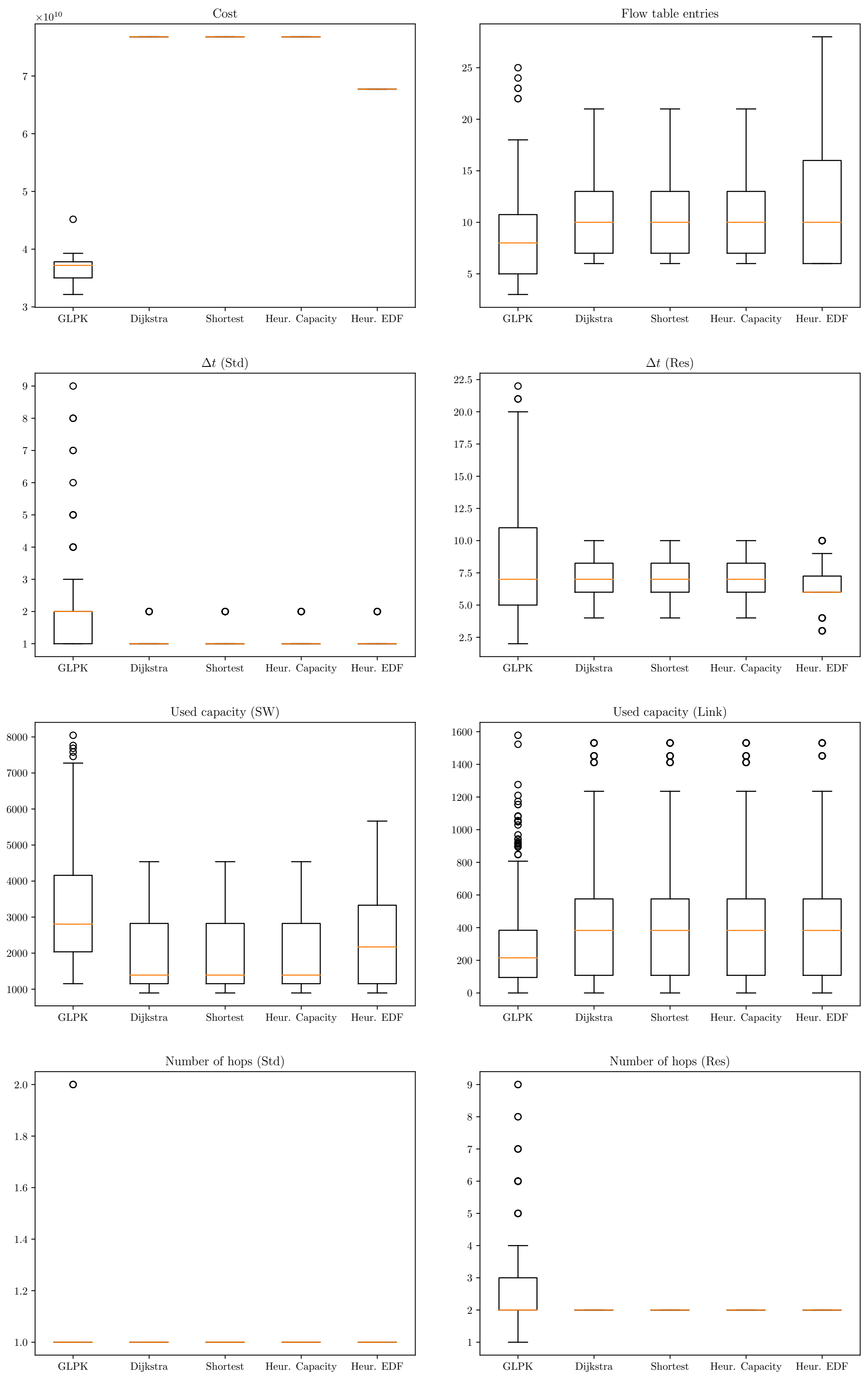


Figure C.1: Results PDH network, adapted from [142]

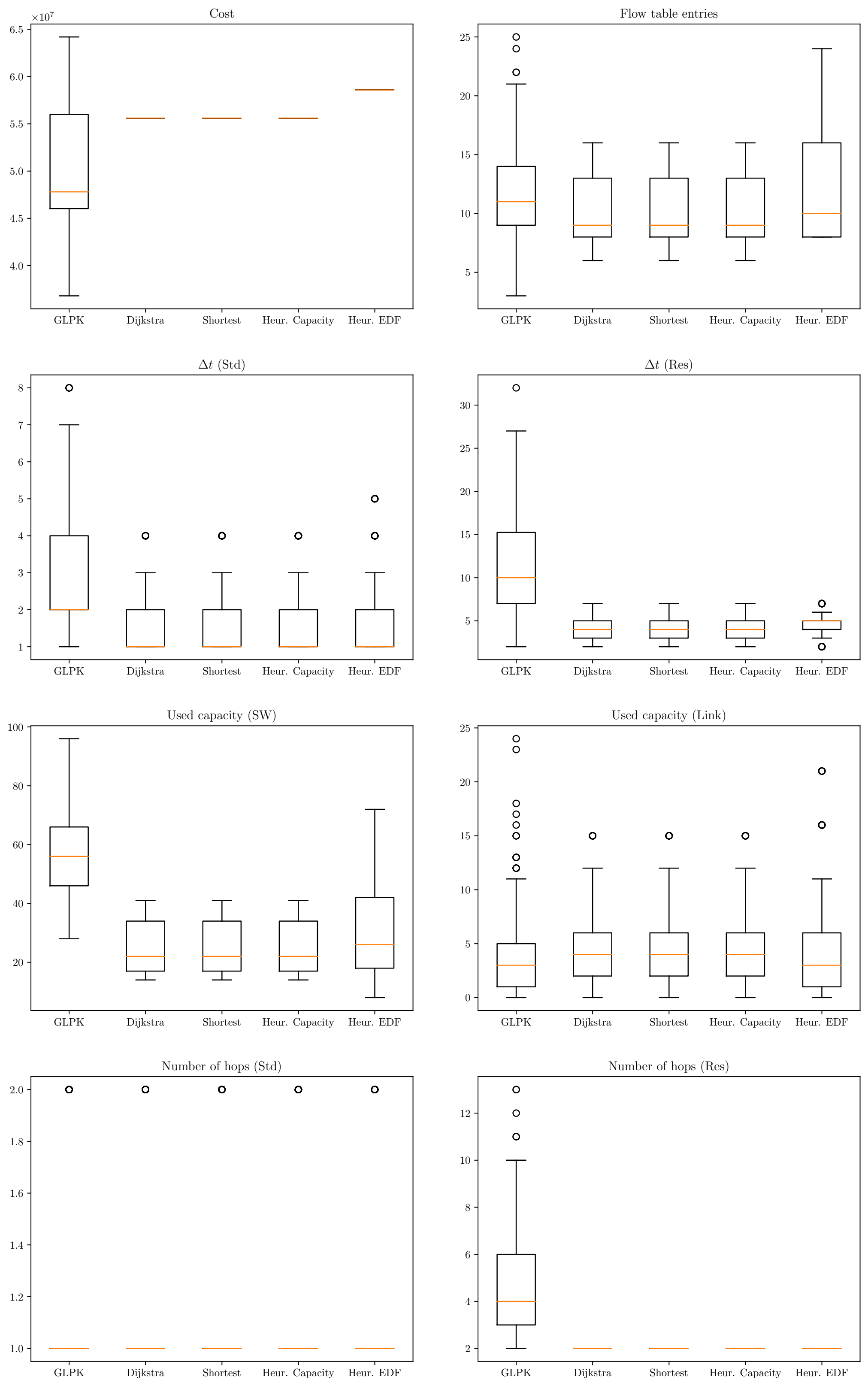


Figure C.2: Results Di-Yuan network, adapted from [142]

Chapter D

List of Acronyms

ACARE	Advisory Council for Aeronautics Research in Europe. A private-public partnership European advisory body for aeronautics.
AD	Anomaly detection.
AFDX	Avionics Full DupleX Switched Ethernet.
ARIMA	Auto-regressive integrated moving average.
ARINC	Aeronautical Radio, Incorporated.
ATC	Air traffic control.
ATM	Air traffic management.
BAG	Bandwidth allocation gap.
BFT	Byzantine fault tolerance.
BGR	Blind geographic routing.
BYOD	Bring your own device. Including user-owned devices into networks rather than providing them by design.
COTS	Commercial off-the shelf.
DDoS	Distributed denial of service.
DFDR	Digital flight data recorder. Mandatory crash- and fire-protected hard drive that records flight data parameters so that an accident flight can be analysed and reconstructed.
DFN	Deutsches Forschungsnetz.
DLT	Distributed ledger technology.
DOS	Denial of service.
EASA	European Aviation Safety Agency.
EDF	Earliest deadline first.
EUROCAE	Time-sensitive network.

FAA	Federal Aviation Administration.
FPGA	Field-programmable gate array.
GLPK	GNU linear programming kit.
HMM	Hidden Markov Models.
ICAO	International Civil Aviation Organization.
ID	Anomaly detection.
IDS	European Aviation Safety Agency.
IFALPA	International Federation of Air Line Pilots' Associations. An international non-profit organisation representing pilots.
IFE	In-flight entertainment. Computer system that provides entertainment to passengers on airliners. Passengers interact with it using little screens attached to their seat.
ILP	Integer linear programming.
IoT	Internet of things. Connecting sensors, controllers etc. to the internet.
ISO	International Organization for Standardization.
ISP	Internet service provider.
IVHM	Integrated vehicle health management.
LP	Linear programming.
MAC	Medium access control.
MAPE	Monitor, analyse, plan, execute.
MPA	Multi-party authorisation.
NMS	Network management system.
OF	OpenFlow. Open communication protocol for SDN.
OS	Operating system.
OSI	Open Systems Interconnection. Reference model for layered network architectures.
PCA	Principal component analysis.
PLC	Programmable logic controller.
QAR	Quick access recorder. Device that records the same data set as a DFDR, but not protected. Used for maintenance purposes.
RAM	Random access memory.
SAN	Stochastic activity network. A graphical high-level language to describe system behaviour.
SCADA	Supervisory control and data acquisition.
SDN	Software-defined networking. Technology that separates the forwarding of data packets from routing.
SVM	Support Vector Machine.
TCAM	Ternary content-addressable memory. Memory that holds forwarding rules in SDN switches.

TCP	Transmission control protocol. Stream-oriented, reliable, transport layer protocol.
TSN	Time-sensitive network.
UDP	User datagram protocol. Datagram-oriented, unreliable transport layer protocol.
UNECE	United Nations Economic Commission for Europe.
V2X	Vehicle to infrastructure. Communication between vehicle and infrastructure such as mobile networks.
VLAN	Virtual local area network.

Bibliography

[1] A. G. Abbasi and Z. Khan. “VeidBlock: Verifiable Identity Using Blockchain and Ledger in a Software Defined Network”. In: *Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*. UCC ’17 Companion. Austin, Texas, USA: ACM, 2017, pp. 173–179. ISBN: 978-1-4503-5195-9. DOI: 10.1145/3147234.3148088.

[2] M. Abu Alsheikh et al. “Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications”. In: *Communications Surveys Tutorials, IEEE* 16.4 (2014), pp. 1996–2018.

[3] ACARE. “Ensuring Safety and Security”. In: *Strategic Research & Innovation Agenda*. Advisory Council for Aviation Research and Innovation in Europe, 2012.

[4] N. van Adrichem. “Resilience and Application Deployment in Software-Defined Networks”. PhD thesis. TU Delft Network Architectures and Services, 2017. DOI: 10.4233/uuid:318d88af-e25e-4a7e-8d37-18770fe980c4.

[5] N. van Adrichem, F. Iqbal, and F. Kuipers. “Computing Backup Forwarding Rules in Software-Defined Networks”. In: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2016, pp. 179–185. DOI: 10.1109/NFV-SDN.2016.7919495.

[6] Aeronautical Radio, Inc. *Aircraft Data Network - Part 5 - Network Domain Characteristics and Interconnection*. Tech. rep. ARINC, 2005.

[7] Aeronautical Radio, Inc. *Avionics Application Standard Software Information*. Tech. rep. ARINC, 2007.

[8] S. Agarwal, M. Kodialam, and T. V. Lakshman. “Traffic Engineering in Software Defined Networks”. In: *2013 Proceedings IEEE INFOCOM*. 2013, pp. 2211–2219. DOI: 10.1109/INFCOM.2013.6567024.

[9] P. Amangele et al. “Hierarchical Machine Learning for IoT Anomaly Detection in SDN”. In: *2019 International Conference on Information Technologies (InfoTech)*. 2019, pp. 1–4.

[10] A. Amouri, V. Alaparthy, and S. Morgera. “Cross Layer-Based Intrusion Detection Based on Network Behavior for IoT”. In: *2018 IEEE 19th Wireless and Microwave Technology Conference (WAMICON)*. 2018, pp. 1–4.

[11] P. Andritsos et al. “LIMBO: Scalable Clustering of Categorical Data”. In: *EDBT*. Springer. 2004, pp. 123–146.

[12] H. Arora, B.K. Mishra, and T.S. Raghu. “Autonomic-Computing Approach to Secure Knowledge Management: A Game-Theoretic Analysis”. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 36.3 (2006), pp. 487–497. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2006.871724.

[13] S. Astaneh and S.S. Heydari. “Multi-Failure Restoration with Minimal Flow Operations in Software Defined Networks”. In: *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*. 2015, pp. 263–266. DOI: 10.1109/DRCN.2015.7149024.

[14] S. Astaneh and S.S. Heydari. “Optimization of SDN Flow Operations in Multi-Failure Restoration Scenarios”. In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 421–432. ISSN: 1932-4537. DOI: 10.1109/TNSM.2016.2580590.

[15] A. Avizienis. “Design of Fault-Tolerant Computers”. In: *Fall Joint Computer Conference*. 1967, pp. 733–743.

[16] S. Balon, F. Skivée, and G. Leduc. “How Well Do Traffic Engineering Objective Functions Meet TE Requirements?” In: *IFIP-TC6 Networking Conference (Networking)*. May 2006.

[17] L.E. Baum et al. “A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains”. In: *The Annals of Mathematical Statistics* 41.1 (1970), pp. 164–171.

[18] R. Bellman. “On a Routing Problem”. In: *Quarterly of Applied Mathematics* 16 (1958), pp. 87–90.

[19] M.H. Bhuyan, D.K. Bhattacharyya, and J.K. Kalita. “Network Anomaly Detection: Methods, Systems and Tools”. In: *Communications Surveys Tutorials, IEEE* 16.1 (2014), pp. 303–336. ISSN: 1553-877X. DOI: 10.1109/SURV.2013.052213.00046.

[20] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press, 1995.

[21] F. Botelho et al. “Design and Implementation of a Consistent Data Store for a Distributed SDN Control Plane”. In: *2016 12th European Dependable Computing Conference (EDCC)*. 2016, pp. 169–180. DOI: 10.1109/EDCC.2016.12.

[22] M. Bouet, K. Phemius, and J. Leguay. “Distributed SDN for Mission-Critical Networks”. In: *2014 IEEE Military Communications Conference*. 2014, pp. 942–948. DOI: 10.1109/MILCOM.2014.162.

[23] S. Budalakoti, A.N. Srivastava, and M.E. Otey. “Anomaly Detection and Diagnosis Algorithms for Discrete Symbol Sequences with Applications to Airline Safety”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 39.1 (2009), pp. 101–113. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2008.2007248.

[24] J. Camacho et al. “Group-Wise Principal Component Analysis for Exploratory Intrusion Detection”. In: *IEEE Access* 7 (2019), pp. 113081–113093.

[25] F. Camci and R.B. Chinnam. “Health-State Estimation and Prognostics in Machining Processes”. In: *Automation Science and Engineering, IEEE Transactions on* 7.3 (2010), pp. 581–597. ISSN: 1545-5955. DOI: 10.1109/TASE.2009.2038170.

[26] M. Canini et al. “Renaissance: A Self-Stabilizing Distributed SDN Control Plane”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 2018, pp. 233–243. DOI: 10.1109/ICDCS.2018.00032.

[27] A. Capone et al. “Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState”. In: *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*. 2015, pp. 25–32. DOI: 10.1109/DRCN.2015.7148981.

[28] B. Cattelan and S. Bondorf. “Iterative Design Space Exploration for Networks Requiring Performance Guarantees”. In: *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. 2017, pp. XX–XX.

[29] W.B. Cavnar, J. M. Trenkle, et al. “N-Gram-Based Text Categorization”. In: *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*. Vol. 161175. 1994.

[30] E.K. Çetinkaya et al. “Modelling Communication Network Challenges for Future internet Resilience, Survivability, and Disruption Tolerance: A Simulation-Based Approach”. In: *Telecommunication Systems* 52.2 (2013), pp. 751–766. ISSN: 1572-9451. DOI: 10.1007/s11235-011-9575-4.

[31] V. Chandola, A. Banerjee, and V. Kumar. “Anomaly Detection for Discrete Sequences: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 823–839.

[32] C. Chang, W. Hsu, and I. Liao. “Anomaly Detection for industrial Control Systems Using K-Means and Convolutional Autoencoder”. In: *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2019, pp. 1–6.

[33] H. Charara et al. “Methods for Bounding End-To-End Delays on An AFDX Network”. In: *18th Euromicro Conference on Real-Time Systems (ECRTS’06).* 2006, 10 pp.–202. DOI: `10.1109/ECRTS.2006.15`.

[34] H. Chen, Y. Chen, and D.H. Summerville. “A Survey on the Application of FPGAs for Network infrastructure Security”. In: *Communications Surveys Tutorials, IEEE* 13.4 (2011), pp. 541–561. ISSN: 1553-877X. DOI: `10.1109/SURV.2011.072210.00075`.

[35] L. Chen and A. Avizienis. “N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation”. In: *8th Symposium on Fault-Tolerant Computing.* Vol. 111. 1978, pp. 3–9. DOI: `10.1109/FTCSH.1995.532621`.

[36] T. Cheng et al. “Evasion Techniques: Sneaking through Your Intrusion Detection/Prevention Systems”. In: *Communications Surveys Tutorials, IEEE* 14.4 (2012), pp. 1011–1020. ISSN: 1553-877X. DOI: `10.1109/SURV.2011.092311.00082`.

[37] A.K. Chorppath, T. Alpcan, and H. Boche. “Bayesian Mechanisms and Detection Methods for Wireless Network with Malicious Users”. In: *IEEE Transactions on Mobile Computing* 15.10 (2016), pp. 2452–2465.

[38] C.S. Collberg and C. Thomborson. “Watermarking, Tamper-Proofing, and Obfuscation-Tools for Software Protection”. In: *Software Engineering, IEEE Transactions on* 28.8 (2002), pp. 735–746.

[39] J. Cámara et al. “Robustness-Driven Resilience Evaluation of Self-Adaptive Software Systems”. In: *IEEE Transactions on Dependable and Secure Computing* 14.1 (2017), pp. 50–64. ISSN: 1545-5971. DOI: `10.1109/TDSC.2015.2429128`.

[40] T. Dang et al. “Trend-Adaptive Multi-Scale PCA for Data Fault Detection in IoT Networks”. In: *2018 International Conference on Information Networking (ICOIN).* 2018, pp. 744–749.

[41] S. Das et al. “Multiple Kernel Learning for Heterogeneous Anomaly Detection: Algorithm and Aviation Safety Case Study”. In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM. 2010, pp. 47–56.

[42] N. de Palma et al. “Self-Protection in a Clustered Distributed System”. In: *Parallel and Distributed Systems, IEEE Transactions on* 23.2 (2012), pp. 330–336. ISSN: 1045-9219. DOI: `10.1109/TPDS.2011.161`.

[43] G.L. Dilingham, G.C. Wilshusen, and N. Barkakati. *FAA Needs a More Comprehensive Approach to Address Cybersecurity as Agency Transitions to NextGen.* Tech. rep. United States Government Accountability Office, 2015.

[44] T.E. Dube et al. “Malware Target Recognition of Unknown Threats”. In: *Systems Journal, IEEE* 7.3 (2013), pp. 467–477. ISSN: 1932-8184. DOI: 10.1109/JSYST.2012.2221913.

[45] P. Duessel et al. “Tracing Privilege Misuse Through Behavioral Anomaly Detection in Geometric Spaces”. In: *2020 13th International Conference on Systematic Approaches to Digital Forensic Engineering (SADFE)*. 2020, pp. 22–31.

[46] I. El Naqa. “Detection and Prediction of Radiotherapy Errors”. In: *Machine Learning in Radiation Oncology: Theory and Applications*. Ed. by I. El Naqa, R. Li, and M.J. Murphy. Springer, 2015.

[47] M.F. Elrawy, A.I. Awad, and H.F.A. Hamed. “Intrusion detection systems for IoT-based smart environments: a survey”. In: *Journal of Cloud Computing* 7.1 (2018), p. 21. ISSN: 2192-113X. DOI: 10.1186/s13677-018-0123-6.

[48] European Aviation Safety Agency. *Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes*. Tech. rep. CS-25 and AMC. https://www.easa.europa.eu/official-publication/certification-specifications, Accessed 11.12.2017. EASA.

[49] European Aviation Safety Agency. *Commission Regulation (EU) No 965/2012 on air operations and related EASA Decisions (AMC & GM and CS-FTL.1)*. Tech. rep. Regulation (EU) 965/2012. https://www.easa.europa.eu/document-library/regulations, Accessed 11.12.2017. EASA.

[50] European Aviation Safety Agency. *What does the Agency not do?* https://www.easa.europa.eu/faq/19225, Accessed 20.03.2021.

[51] L. Fawcett et al. “Tennison: A Distributed SDN Framework for Scalable Network Security”. In: *IEEE Journal on Selected Areas in Communications* 36.12 (2018), pp. 2805–2818. ISSN: 0733-8716. DOI: 10.1109/JSAC.2018.2871313.

[52] Federal Aviation Administration. *FAA Strategic Plan FY 2019–2022*. Tech. rep. FAA, 2019.

[53] Federal Aviation Administration. *Special Conditions: Boeing Model 787-8 Airplane; Systems and Data Networks Security-Protection of Airplane Systems and Data Networks from Unauthorized External Access*. Tech. rep. https://www.federalregister.gov/documents/2007/12/28/E7-25075/special-conditions-boeing-model-787-8-airplane-systems-and-data-networks-security-protection-of, Accessed 20.03.2021. FAA.

[54] G. Fernandes Jr et al. “Network Anomaly Detection Using IP Flows with Principal Component Analysis and Ant Colony Optimization”. In: *Journal of Network and Computer Applications* 64 (2016), pp. 1–11.

[55] P. Fonseca et al. “A Replication Component for Resilient OpenFlow-Based Networking”. In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 933–939. DOI: 10.1109/NOMS.2012.6212011.

[56] G.D. Forney. “The Viterbi Algorithm”. In: *Proceedings of the IEEE* 61.3 (1973), pp. 268–278.

[57] B. fortz and M. Thorup. “Internet Traffic Engineering by Optimizing OSPF Weights”. In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Vol. 2. 2000, 519–528 vol.2. DOI: 10.1109/INFCOM.2000.832225.

[58] S.R. Gaddam, V.V. Phoha, and K.S. Balagani. “K-Means+ID3: A Novel Method for Supervised Anomaly Detection by Cascading K-Means Clustering and ID3 Decision Tree Learning Methods”. In: *Knowledge and Data Engineering, IEEE Transactions on* 19.3 (2007), pp. 345–354. ISSN: 1041-4347. DOI: 10.1109/TKDE.2007.44.

[59] F. Geyer. “End-to-End Flow-Level Quality-of-Service Guarantees for Switched Networks”. PhD thesis. Technical University of Munich, 2015.

[60] M. Gharib et al. “Dealing with Functional Safety Requirements for Automotive Systems: A Cyber-Physical-Social Approach”. In: *Critical Information infrastructures Security*. Ed. by G. D’Agostino and A. Scala. Cham: Springer International Publishing, 2018, pp. 194–206. ISBN: 978-3-319-99843-5.

[61] U. Ghosh et al. “A Simulation Study on Smart Grid Resilience Under Software-Defined Networking Controller Failures”. In: *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*. CPSS ’16. Xi’an, China: ACM, 2016, pp. 52–58. ISBN: 978-1-4503-4288-9. DOI: 10.1145/2899015.2899020.

[62] GNU Linear Programming Kit. https://www.gnu.org/software/glpk, Accessed 20.03.2021.

[63] G. Gonzalez-Granadillo et al. “Towards an Automated and Dynamic Risk Management Response System”. In: *Sec. IT Sys.* Ed. by Billy Bob Brumley and Juha Röning. Cham: Springer International Publishing, 2016, pp. 37–53. ISBN: 978-3-319-47560-8.

[64] G. Gonzalez Granadillo et al. “Selection of Mitigation Actions Based on Financial and Operational Impact Assessments”. In: *11th International Conference on Availability, Reliability and Security (ARES)*. 2016.

[65] N. Gray et al. “Evaluation of a Distributed Control Plane for Managing Heterogeneous SDN-enabled and Legacy Networks”. In: *2018 IEEE Seventh International Conference on Communications and Electronics (ICCE)*. 2018, pp. 361–366. DOI: 10.1109/CCE.2018.8465724.

[66] C. Grosan, A. Abraham, and B. Helvik. “Building Multiobjective Resilient Networks (Invited Paper)”. In: May 2008, pp. 204–209. ISBN: 0-7695-3114-8. DOI: 10.1109/UKSIM.2008.31.

[67] Gurobi Optimization Inc. *Gurobi Optimizer Reference Manual.* https://www.gurobi.com, Accessed 20.03.2021.

[68] A. Hadri, K. Chougdali, and R. Touahni. “A Network Intrusion Detection Based on Improved Nonlinear Fuzzy Robust PCA”. In: *2018 IEEE 5th International Congress on Information Science and Technology (CiSt).* 2018, pp. 636–641.

[69] S. Hangal and M.S. Lam. “Tracking Down Software Bugs Using Automatic Anomaly Detection”. In: *Proceedings of the 24th International Conference on Software Engineering.* ACM. 2002, pp. 291–301.

[70] M. Hartmann et al. “Objective Functions for Optimization of Resilient and Non-Resilient IP Routing”. In: *2009 7th International Workshop on Design of Reliable Communication Networks.* 2009, pp. 289–296. DOI: 10.1109/DRCN.2009.5339993.

[71] N. Heard and P. Rubin-Delanchy. “Network-Wide Anomaly Detection Via the Dirichlet Process”. In: *2016 IEEE Conference on Intelligence and Security Informatics (ISI).* 2016, pp. 220–224.

[72] P. Heise, F. Geyer, and R. Obermaisser. “Self-Configuring Deterministic Network with in-Band Configuration Channel”. In: *2017 Fourth International Conference on Software Defined Systems (SDS).* 2017, pp. 162–167. DOI: 10.1109/SDS.2017.7939158.

[73] J.L. Hellerstein et al. *Feedback Control of Computing Systems.* John Wiley & Sons, 2004.

[74] V. Heorhiadi, M.K. Reiter, and V. Sekar. “Accelerating the Development of Software-Defined Network Optimization Applications Using SOL”. In: *CoRR* abs/1504.07704 (2015). https://dblp.uni-trier.de/rec/bib/journals/corr/HeorhiadiRS15, Accessed 02.05.2017.

[75] N. Herold et al. “An Optimal Metric-Aware Response Selection Strategy for Intrusion Response Systems”. In: *Foundations and Practice of Security: 8th International Symposium, FPS 2016, Quebec, Canada, October 24-26, 2016.* Springer International Publishing, 2016.

[76] H. Hijazi, P. Bonami, and A. Ouorou. “Robust Delay-Constrained Routing in Telecommunications”. In: *Annals of Operations Research* 206.1 (2013), pp. 163–181. ISSN: 1572-9338. DOI: 10.1007/s10479-013-1371-y.

[77] H. Huang et al. “Joint Optimization of Rule Placement and Traffic Engineering for QoS Provisioning in Software Defined Network”. In: *IEEE Transactions on*

Computers 64.12 (2015), pp. 3488–3499. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2401031.

[78] T. Hurley, J.E. Perdomo, and A. Perez-Pons. “HMM-Based Intrusion Detection System for Software Defined Networking”. In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2016, pp. 617–621.

[79] ICAO. *Annex 17 to the Convention on International Civil Aviation: Security. Safeguarding International Civil Aviation Against Acts of Unlawful Interference.* Tech. rep. Annex 17. ICAO, 2017.

[80] International Air Transport Association. *Position Paper on Cybersecurity.* 2015.

[81] International Federation of Air Line Pilots’ Associations. *Cyber Threats: Who Controls Your Aircraft?* Position paper. 2013.

[82] International Standardization Organization. *Road Vehicles – Cybersecurity Engineering.* Tech. rep. ISO/SAE DIS 21434. ISO, 2020.

[83] R. Isermann. *Fault-Diagnosis Systems.* Springer, 2006.

[84] ISO. *Information Technology – Programming Languages – Ada.* Tech. rep. 8652:2012. ISO, 2012.

[85] ISO. *Road Vehicles – Functional Safety – Part 2: Management of Functional Safety.* Tech. rep. 26262-2. ISO, 2018.

[86] H. Izakian and W. Pedrycz. “Anomaly Detection and Characterization in Spatial Time Series Data: A Cluster-Centric Approach”. In: *Fuzzy Systems, IEEE Transactions on* 22.6 (2014), pp. 1612–1624. ISSN: 1063-6706. DOI: 10.1109/TFUZZ.2014.2302456.

[87] X. Jin et al. “Anomaly Detection in Nuclear Power Plants via Symbolic Dynamic Filtering”. In: *Nuclear Science, IEEE Transactions on* 58.1 (2011), pp. 277–288. ISSN: 0018-9499. DOI: 10.1109/TNS.2010.2088138.

[88] O. Kabadurmus and A.E. Smith. “Evaluating Reliability/Survivability of Capacitated Wireless Networks”. In: *IEEE Transactions on Reliability* 67.1 (2018), pp. 26–40. ISSN: 0018-9529. DOI: 10.1109/TR.2017.2712667.

[89] N. Kang et al. “Optimizing the One Big Switch Abstraction in Software-Defined Networks”. In: *Proc. 8th Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT)*. 2013.

[90] Y. Kanizo, D. Hay, and I. Keslassy. “Palette: Distributing Tables in Software-Defined Networks”. In: *2013 Proc. IEEE INFOCOM.* 2013, pp. 545–549.

[91] P.G. Kannan et al. “Raptor: Scalable Rule Placement over Multiple Path in Software Defined Networks”. In: *16th Int. IFIP TC6 Netw. Conf.* 2017.

[92] S. Karnouskos. “Stuxnet Worm Impact on industrial Cyber-Physical System Security”. In: *IECON 2011 - 37th Annual Conference of the IEEE industrial Electronics Society.* 2011, pp. 4490–4494. DOI: `10.1109/IECON.2011.6120048`.

[93] M. Karthi and K. Sakthipriya. “Self-Protect Computing Systems Towards Model-Based Validated Autonomic Approach”. In: *International Journal of Scientific & Engineering Research* 6.3 (2015), pp. 302–306.

[94] H. Kasai, W. Kellerer, and M. Kleinsteuber. “Network Volume Anomaly Detection and Identification in Large-Scale Networks Based on Online Time-Structured Traffic Tensor Tracking”. In: *IEEE Transactions on Network and Service Management* 13.3 (2016), pp. 636–650.

[95] J.O. Kephart and D.M. Chess. “The Vision of Autonomic Computing”. In: *Computer* 36 (2003), pp. 41–50.

[96] W. Khreich et al. “An Anomaly Detection System Based on Variable N-Gram Features and One-Class SVM”. In: *Information and Software Technology* 91 (2017), pp. 186–197.

[97] C. Kiennert et al. “A Survey on Game-Theoretic Approaches for Intrusion Detection and Response Optimization”. In: *ACM Comput. Surv.* 51.5 (Aug. 2018), 90:1–90:31. ISSN: 0360-0300. DOI: `10.1145/3232848`.

[98] C. Kolbitsch, E. Kirda, and C. Kruegel. “The Power of Procrastination: Detection and Mitigation of Execution-Stalling Malicious Code”. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security.* ACM. 2011, pp. 285–296.

[99] R. Kölle, G. Markarian, and A. Tartar. *Aviation Security Engineering: A Holistic Approach.* Artech House, 2011.

[100] S.B. Kotsiantis, I. Zaharakis, and P. Pintelas. “Supervised Machine Learning: A Review of Classification Techniques”. In: *Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, information Retrieval and Pervasive Technologies.* Ed. by I.G. Maglogiannis et al. Frontiers in artificial intelligence and applications. IOS Press, 2007.

[101] P. Kotzanikolaou and C. Douligeris. “Computer Network Security: Basic Background and Current Issues”. In: *Network Security.* John Wiley & Sons, Ltd, 2006. Chap. 1, pp. 1–12. ISBN: 9780470099742. DOI: `10.1002/9780470099742.ch1`.

[102] T. Kovanen, G. David, and T. Hämäläinen. “Survey: Intrusion Detection Systems in Encrypted Traffic”. In: *Internet of Things, Smart Spaces, and Next Generation Networks and Systems.* Ed. by Olga Galinina, Sergey Balandin, and Yevgeni Koucheryavy. Cham: Springer International Publishing, 2016, pp. 281–293. ISBN: 978-3-319-46301-8.

[103] D. Kreutz et al. “Software-Defined Networking: A Comprehensive Survey”. In: *Proceedings of the IEEE* 103.1 (2015), pp. 14–76. ISSN: 0018-9219. DOI: 10.1109/JPROC.2014.2371999.

[104] M. Kuzniar et al. “Automatic Failure Recovery for Software-Defined Networks”. In: *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)* (2013).

[105] A. Lakhina, M. Crovella, and C. Diot. “Diagnosing Network-Wide Traffic Anomalies”. In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 219–230.

[106] J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queing Systems for the Internet.* Vol. 2050. Lecture Notes in Computer Science. Springer, 2001. DOI: 10.1007/3-540-45318-0.

[107] T. Leckie and A. Yasinsac. “Metadata for Anomaly-Based Security Protocol Attack Deduction”. In: *Knowledge and Data Engineering, IEEE Transactions on* 16.9 (2004), pp. 1157–1168. ISSN: 1041-4347. DOI: 10.1109/TKDE.2004.43.

[108] W. Lin. “Secure Multi-Party Authorization in Clouds”. https://etd.ohiolink.edu/!etd.send_file?accession=osu1429041745, Accessed 06.03.2018. PhD thesis. The Ohio State University.

[109] R.P. Lippmann. “An Introduction To Computing with Neural Nets”. In: *ASSP Magazine, IEEE* 4.2 (1987), pp. 4–22.

[110] P. Liu and L. Li. *A Game Theoretic Approach to Attack Prediction.* Tech. rep. PSU-S2-2002-001. Penn State Cyber Security Group, 2002.

[111] Y. Liu and S. Chawla. “Social Media Anomaly Detection: Challenges and Solutions”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM. 2015, pp. 2317–2318.

[112] Y. Liu, J. Lv, and S. Ma. “A Real Time Anomaly Detection Method Based on Variable N-Gram for Flight Data”. In: *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS).* 2018, pp. 370–376.

[113] M. Luo et al. “An Adaptive Multi-Path Computation Framework for Centrally Controlled Networks”. In: *Computer Networks* 83 (2015), pp. 30 –44. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2015.02.004.

[114] R.E. Lyons and W. Vanderkulk. “The Use of Triple-Modular Redundancy to Improve Computer Reliability”. In: *IBM J. Res. Dev.* 6.2 (Apr. 1962), pp. 200–209. ISSN: 0018-8646. DOI: 10.1147/rd.62.0200.

[115] G. Macher et al. “Automotive SPICE, Safety and Cybersecurity Integration”. In: *Computer Safety, Reliability, and Security.* Ed. by S. Tonetta, E. Schoitsch, and

F.n Bitsch. Cham: Springer International Publishing, 2017, pp. 273–285. ISBN: 978-3-319-66284-8.

[116] C. Mas Machuca et al. “Technology-Related Disasters: A Survey Towards Disaster-Resilient Software Defined Networks”. In: *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*. 2016, pp. 35–42. DOI: 10.1109/RNDM.2016.7608265.

[117] A. Malik, B. Aziz, and M. Bader-El-Den. “Finding Most Reliable Paths for Software Defined Networks”. In: *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2017, pp. 1309–1314. DOI: 10.1109/IWCMC.2017.7986474.

[118] C. Manikopoulos and S. Papavassiliou. “Network Intrusion and Fault Detection: A Statistical Anomaly Approach”. In: *Communications Magazine, IEEE* 40.10 (2002), pp. 76–82. ISSN: 0163-6804. DOI: 10.1109/MCOM.2002.1039860.

[119] H. Mao et al. “Resource Management with Deep Reinforcement Learning”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM. 2016, pp. 50–56.

[120] D.J. Marchette. “A Statistical Method for Profiling Network Traffic.” In: *Workshop on Intrusion Detection and Network Monitoring*. 1999, pp. 119–128.

[121] H. Marzi. “Real-Time Fault Detection and Isolation in Industrial Machines Using Learning Vector Quantization”. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 218.8 (2004), pp. 949–959. DOI: 10.1243/0954405041486109.

[122] S. Mascaro, A.E. Nicholso, and K.B. Korb. “Anomaly Detection in Vessel Tracks Using Bayesian Networks”. In: *International Journal of Approximate Reasoning* 55.1, Part 1 (2014). Applications of Bayesian Networks, pp. 84 –98. ISSN: 0888-613X. DOI: https://dx.doi.org/10.1016/j.ijar.2013.03.012.

[123] Mininet. https://mininet.org/, Accessed 20.03.2021.

[124] R. Mitchell and I.-R. Chen. “A Survey of Intrusion Detection Techniques for Cyber-Physical Systems”. In: *ACM Comput. Surv.* 46.4 (Mar. 2014), 55:1–55:29. ISSN: 0360-0300. DOI: 10.1145/2542049.

[125] P.M. Mohan, T. Truong-Huu, and M. Gurusamy. “Fault Tolerance in TCAM-Limited Software Defined Networks”. In: *Computer Networks* 116 (2017), pp. 47 –62. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2017.02.009.

[126] E. Moore and C. Shannon. “Reliable Circuits Using Less Reliable Relays”. In: *Journal of the Franklin institute* 262.3 (1956), pp. 191–208.

[127] G. Münz, S. Li, and G. Carle. “Traffic Anomaly Detection Using k-Means Clustering”. In: *Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, 4. GI/ITG-Workshop MMBnet*. 2007.

[128] Y.L. Murphey et al. “Model-Based Fault Diagnosis in Electric Drives Using Machine Learning”. In: *IEEE/ASME Transactions on Mechatronics* 11.3 (2006), pp. 290–303. ISSN: 1083-4435. DOI: `10.1109/TMECH.2006.875568`.

[129] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System.* `https://bitcoin.org/bitcoin.pdf`, Accessed 23.02.2018.

[130] H. Natarajan, S.K. Krause, and H.L. Gradstein. *Distributed Ledger Technology (DLT) and Blockchain.* `https://documents.worldbank.org/curated/en/177911513714062215/pdf/122140-WP-PUBLIC-Distributed-Ledger-Technology-and-Blockchain-Fintech-Notes.pdf`, Accessed 23.02.2018. Washington, D.C., USA.

[131] S. Ntalampiras. “Fault Identification in Distributed Sensor Networks Based on Universal Probabilistic Modeling”. In: *Neural Networks and Learning Systems, IEEE Transactions on* 26.9 (2015), pp. 1939–1949. ISSN: 2162-237X. DOI: `10.1109/TNNLS.2014.2362015`.

[132] P. O’Kane, S. Sezer, and K. McLaughlin. “Obfuscation: the Hidden Malware”. In: *Security Privacy, IEEE* 9.5 (2011), pp. 41–47. ISSN: 1540-7993. DOI: `10.1109/MSP.2011.98`.

[133] Open Networking Foundation. *OpenFlow Switch Specification.* `https://opennetworking.wpengine.com/wp-content/uploads/2014/10/openflow-spec-v1.3.3.pdf`, Accessed 20.03.2021. 2013.

[134] S. Orlowski et al. “SNDlib 1.0 – Survivable Network Design Library”. In: *Networks* 55.3 (2010), pp. 276–286. ISSN: 1097-0037. DOI: `10.1002/net.20371`.

[135] N. Oza, J.P. Castle, and J. Stutz. “Classification of Aeronautics System Health and Safety Documents”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 39.6 (2009), pp. 670–680. ISSN: 1094-6977. DOI: `10.1109/TSMCC.2009.2020788`.

[136] M. Ozay et al. “Machine Learning Methods for Attack Detection in the Smart Grid”. In: *IEEE Transactions on Neural Networks and Learning Systems* 27.8 (2016), pp. 1773–1786.

[137] D. Papadimitriou, D. Colle, and P. Demeester. “Mixed-Integer Optimization for the Combined Capacitated Facility Location-Routing Problem”. In: *2016 12th International Conference on the Design of Reliable Communication Networks (DRCN).* 2016, pp. 14–22. DOI: `10.1109/DRCN.2016.7470830`.

[138] D. Parikh and T. Chen. “Data Fusion and Cost Minimization for Intrusion Detection”. In: *Information forensics and Security, IEEE Transactions on* 3.3 (2008), pp. 381–389. ISSN: 1556-6013. DOI: `10.1109/TIFS.2008.928539`.

[139] S. Paris, G.S. Paschos, and J. Leguay. “Dynamic Control for Failure Recovery and Flow Reconfiguration in SDN”. In: *2016 12th International Conference on*

the Design of Reliable Communication Networks (DRCN). 2016, pp. 152–159. DOI: 10.1109/DRCN.2016.7470850.

[140] I.C. Paschalidis and Chen Y. “Anomaly Detection in Sensor Networks Based on Large Deviations of Markov Chain Models”. In: *2008 47th IEEE Conference on Decision and Control*. 2008, pp. 2338–2343.

[141] A. Paverd, A. Martin, and I. Brown. “Security and Privacy in Smart Grid Demand Response Systems”. In: *Smart Grid Security*. Ed. by J. Cuellar. Cham: Springer International Publishing, 2014, pp. 1–15. ISBN: 978-3-319-10329-7.

[142] C. Perner. “Network Optimization for Safety-Critical Systems Using Software-Defined Networks”. In: *Architecture Comput. Sys. (ARCS) 2018*. Ed. by Mladen Berekovic et al. Cham: Springer International Publishing, 2018, pp. 127–138. ISBN: 978-3-319-77610-1.

[143] C. Perner and G. Carle. “Comparison of Optimization Goals for Resilient Routing”. In: *2019 IEEE International Conference on Communications Workshops (ICC Workshops): the 2nd International Workshop on 5G and Cooperative Autonomous Driving (5G Auto) (ICC 2019 Workshop - 5G Auto)*. Shanghai, P.R. China, May 2019.

[144] C. Perner, H. Kinkelin, and G. Carle. “Adaptive Network Management for Safety-Critical Systems”. In: *IM 2019 - IEEE/IFIP Workshop Dissect 2019*. Washington D.C., USA, Apr. 2019.

[145] C. Perner, C. Schmitt, and G. Carle. “Dynamic Network Reconfiguration in Safety-Critical Aeronautical Systems”. In: *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pp. 1–8. DOI: 10.1109/DASC50938.2020.9256497.

[146] T. Pfeiffenberger et al. “Reliable and Flexible Communications for Power Systems: Fault-tolerant Multicast with SDN/OpenFlow”. In: *7th IFIP on New Technologies, Mobility and Security (NTMS)*. 2015.

[147] S. Prabhu et al. “Let Me Rephrase That: Transparent Optimization in SDNs”. In: *Proceedings of the Symposium on SDN Research*. SOSR ’17. Santa Clara, CA, USA: ACM, 2017, pp. 41–47. ISBN: 978-1-4503-4947-5. DOI: 10.1145/3050220.3050226.

[148] Publications Office. *Regulation (EU) 2019/2144 on Type-Approval Requirements for Motor Vehicles*. Tech. rep. Regulation (EU) 2019/2144. European Parliament and Council, 2016.

[149] S. Rajasegarar et al. “Quarter Sphere Based Distributed Anomaly Detection in Wireless Sensor Networks”. In: *Communications, 2007. ICC ’07. IEEE International Conference on*. 2007, pp. 3864–3869. DOI: 10.1109/ICC.2007.637.

[150] B. Randell. “System Structure for Software Fault Tolerance”. In: *Proceedings of the International Conference on Reliable Software.* 1975, pp. 437–449.

[151] E. Sakic and W. Kellerer. “Response Time and Availability Study of RAFT Consensus in Distributed SDN Control Plane”. In: *IEEE Transactions on Network and Service Management* 15.1 (2018), pp. 304–318. ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2775061.

[152] E. Sakic, N. Ðerić, and W. Kellerer. “MORPH: An Adaptive Framework for Efficient and Byzantine Fault-Tolerant SDN Control Plane”. In: *IEEE Journal on Selected Areas in Communications* 36.10 (2018), pp. 2158–2174. ISSN: 0733-8716. DOI: 10.1109/JSAC.2018.2869938.

[153] K. Sampigethaya. “Software-Defined Networking in Aviation: Opportunities and Challenges”. In: *Integrated Communication, Navigation, and Surveillance Conference (ICNS), 2015.* 2015, pp. 1–21. DOI: 10.1109/ICNSURV.2015.7121310.

[154] C. Sample and K. Schaffer. “An Overview of Anomaly Detection”. In: *IT Professional* 15.1 (2013), pp. 8–11. ISSN: 1520-9202. DOI: 10.1109/MITP.2013.7.

[155] A. Schaeffer-Filho et al. “A Framework for the Design and Evaluation of Network Resilience Management”. In: *2012 IEEE Network Operations and Management Symposium.* 2012, pp. 401–408. DOI: 10.1109/NOMS.2012.6211924.

[156] S. Schriegel, T. Kobzan, and J. Jasperneite. “Investigation on a Distributed SDN Control Plane Architecture for Heterogeneous Time Sensitive Networks”. In: *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS).* 2018, pp. 1–10. DOI: 10.1109/WFCS.2018.8402356.

[157] S. Scott-Hayward, G. O’Callaghan, and S. Sezer. “SDN Security: A Survey”. In: *2013 IEEE SDN for Future Networks and Services (SDN4FNS).* 2013, pp. 1–7. DOI: 10.1109/SDN4FNS.2013.6702553.

[158] F. Simmross-Wattenberg et al. “Anomaly Detection in Network Traffic Based on Statistical inference and α-Stable Modeling”. In: *Dependable and Secure Computing, IEEE Transactions on* 8.4 (2011), pp. 494–509. ISSN: 1545-5971. DOI: 10.1109/TDSC.2011.14.

[159] Single European Sky ATM Research. *Study Launched to Address Cyber-Security in SESAR.* https://www.sesarju.eu/index.php/newsroom/all-news/cyber-security-study, Accessed 20.03.2021.

[160] J. Smith. “Certification of on-line learning neural networks”. In: *Proceedings of ASC 2003* (2003).

[161] P. Smith et al. “Management Patterns: SDN-Enabled Network Resilience Management”. In: *2014 IEEE Network Operations and Management Symposium (NOMS).* 2014, pp. 1–9. DOI: 10.1109/NOMS.2014.6838323.

[162] P. Smith et al. “Network Resilience: A Systematic Approach”. In: *IEEE Communications Magazine* 49.7 (2011), pp. 88–97. ISSN: 0163-6804. DOI: 10.1109/MCOM.2011.5936160.

[163] R. Smith, C. Estan, and S. Jha. “Backtracking Algorithmic Complexity Attacks Against a NIDS”. In: *Computer Security Applications Conference, 2006. ACSAC’06. 22nd Annual*. IEEE. 2006, pp. 89–98.

[164] A.A. Sodemann, M.P. Ross, and B.J. Borghetti. “A Review of Anomaly Detection in Automated Surveillance”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 42.6 (2012), pp. 1257–1272. ISSN: 1094-6977. DOI: 10.1109/TSMCC.2012.2215319.

[165] V.A. Sotiris, P.W. Tse, and M.G. Pecht. “Anomaly Detection Through a Bayesian Support Vector Machine”. In: *Reliability, IEEE Transactions on* 59.2 (2010), pp. 277–286. ISSN: 0018-9529. DOI: 10.1109/TR.2010.2048740.

[166] B. Stephens et al. “PAST: Scalable Ethernet for Data Centers”. In: *Proc. 8th Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT)*. 2012, pp. 49–60.

[167] J.P.G. Sterbenz et al. “Evaluation of Network Resilience, Survivability, and Disruption Tolerance: Analysis, Topology Generation, Simulation, and Experimentation”. In: *Telecommunication systems* 52.2 (2013), pp. 705–736. DOI: 10.1007/s11235-011-9573-6.

[168] J.P.G Sterbenz et al. “Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines”. In: *Computer Networks* 54.8 (2010), pp. 1245–1265.

[169] R. Sterritt and D. Bustard. “Towards An Autonomic Computing Environment”. In: *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA’03)*. IEEE. 2003, pp. 699–703.

[170] C. Ten, J. Hong, and C. Liu. “Anomaly Detection for Cybersecurity of the Substations”. In: *Smart Grid, IEEE Transactions on* 2.4 (2011), pp. 865–873. ISSN: 1949-3053. DOI: 10.1109/TSG.2011.2159406.

[171] C. Ten, G. Manimaran, and C. Liu. “Cybersecurity for Critical Infrastructures: Attack and Defense Modeling”. In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 40.4 (2010), pp. 853–865. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2010.2048028.

[172] N. Thanthry and R. Pendse. “Aviation Data Networks: Security Issues and Network Architecture”. In: *Aerospace and Electronic Systems Magazine, IEEE* 20.6 (2005), pp. 3–8. ISSN: 0885-8985. DOI: 10.1109/MAES.2005.1453803.

[173] *UN Regulations on Cybersecurity and Software Updates to Pave the Way for Mass Roll Out of ’Connected Vehicles’*. https://unece.org/press/un-

`regulations-cybersecurity-and-software-updates-pave-way-mass-roll-out-connected-vehicles`, Accessed 22.03.2021.

[174] United States Government Accountability Office. *Aviation Cybersecurity. FAA Should Fully Implement Key Practices to Strengthen Its Oversight of Avionics Risks.* Tech. rep. GAO–21–86. GAO, 2020.

[175] M. Usman, V. Muthukkumarasamy, and X. Wu. "Mobile Agent-Based Cross-Layer Anomaly Detection in Smart Home Sensor Networks Using Fuzzy Logic". In: *Consumer Electronics, IEEE Transactions on* 61.2 (2015), pp. 197–205. ISSN: 0098-3063. DOI: `10.1109/TCE.2015.7150594`.

[176] H.Z. Wang et al. "A Secure and High-Performance Multi-Controller Architecture for Software-Defined Networking". In: *Frontiers of Information Technology & Electronic Engineering* 17.7 (2016), pp. 634–646. ISSN: 2095-9230. DOI: `10.1631/FITEE.1500321`.

[177] Y. Wang and S. Chen. "Safety-Aware Semi-Supervised Classification". In: *Neural Networks and Learning Systems, IEEE Transactions on* 24.11 (2013), pp. 1763–1772. ISSN: 2162-237X. DOI: `10.1109/TNNLS.2013.2263512`.

[178] T. Watanabe et al. "ResilientFlow: Deployments of Distributed Control Channel Maintenance Modules To Recover SDN From Unexpected Failures". In: *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN).* 2015, pp. 211–218. DOI: `10.1109/DRCN.2015.7149015`.

[179] C. Wilkinson. *Obsolescence and Life Cycle Management for Avionics.* Tech. rep. Research performed by Honeywell Aerospace. Federal Aviation Authority, 2013.

[180] Working Party on Automated/Autonomous and Connected Vehicles. *UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regard To Cyber Security and of their Cybersecurity Management Systems.* Tech. rep. United Nations Economic and Social Council, 2020.

[181] C. Wressnegger, Daniel Schwenk G.and Arp, and Konrad Rieck. "A Close Look on n-Grams in Intrusion Detection: Anomaly Detection vs. Classification". In: *Proceedings of the 2013 ACM Workshop on Artificial intelligence and Security.* AISec '13. Berlin, Germany: Association for Computing Machinery, 2013, 67–76. ISBN: 9781450324885. DOI: `10.1145/2517312.2517316`.

[182] D. Wu et al. "LSTM Learning with Bayesian and Gaussian Processing for Anomaly Detection in Industrial IoT". In: *IEEE Transactions on Industrial Informatics* 16.8 (2020), pp. 5244–5253.

[183] N. Ye. "A Markov Chain Model of Temporal Behavior for Anomaly Detection". In: *In Proceedings of the 2000 IEEE Workshop on Information Assurance and Security.* 2000, pp. 171–174.

[184] J.Y. Yen. “An Algorithm for Finding Shortest Routes From All Source Nodes To A Given Destination in General Networks”. In: *Quart. Appl. Math.* 27 (1969/1970), pp. 526–530.

[185] N. Yodo, P. Wang, and Z. Zhou. “Predictive Resilience Analysis of Complex Systems Using Dynamic Bayesian Networks”. In: *IEEE Transactions on Reliability* 66.3 (2017), pp. 761–770. ISSN: 0018-9529. DOI: 10.1109/TR.2017.2722471.

[186] M. Zaman and C. Lung. “Evaluation of Machine Learning Techniques for Network Intrusion Detection”. In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium.* 2018, pp. 1–5. DOI: 10.1109/NOMS.2018.8406212.

[187] J. Zhang, J. Cao, and F. Gao. “Fault Diagnosis for Multivariable Non-Linear Systems Based on Non-Linear Spectrum Feature”. In: *Transactions of the Institute of Measurement and Control* (2016). DOI: 10.1177/0142331215625766.

[188] S. Zhang et al. “An Adaptable Rule Placement for Software-Defined Networks”. In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* 2014, pp. 88–99. DOI: 10.1109/DSN.2014.24.

[189] C. Zhou et al. “Design and Analysis of Multimodel-Based Anomaly Intrusion Detection Systems in Industrial Process Automation”. In: *Systems, Man, and Cybernetics: Systems, IEEE Transactions on* (2015). ISSN: 2168-2216. DOI: 10.1109/TSMC.2015.2415763.