Real-Time Guarantees, Dependability and Self-Configuration in Future Avionic Networks

DISSERTATION
zur Erlangung des Grades eines Doktors
der Ingenieurwissenschaften

vorgelegt von
M.Sc. M.Sc. M.Sc. Peter Heise

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen 2018

Betreuer und erster Gutachter:
Prof. Dr.-Ing. habil. Roman Obermaisser
Universität Siegen

Zweiter Gutachter:
Prof. Dr. David Hausheer
Universität Magdeburg

Tag der mündlichen Prüfung: 21. März 2018

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

Abstract

Many of today's applications are in need of a real-time communication infrastructure. Several commercially available solutions have been designed for this purpose, each with a different focus. To achieve low cost network services, a standardized solution featuring Ethernet compliance is currently being developed by the IEEE *Time-Sensitive Networking Task Group* (TSN). The main features of this standard are seamless redundancy through multiple disjunctive paths, low latency for high priority traffic through frame-preemption and improved clock synchronization mechanisms as compared to existing standards. While the suitability of the mentioned standard for avionic purposes still has to be evaluated, neither of the standards help when deciding where to route which traffic or how to dimension the network in order to achieve optimal traffic and application distribution.

Similarly, in avionics a trend towards a more modular and decentralized cabin infrastructure can be seen, potentially improving reliability by having a higher fault tolerance and the removal of central servers to save weight. With more degrees of freedom, however, new means for configuring such a network are needed to speed up the assembly process. While topology discovery and address configuration algorithms are widely available, little focus has been paid to the special requirements needed in the avionic domain, like real-time behavior, fault isolation and seamless redundancy. In such a fully dynamic cabin, a cloud like service can be envisioned that allows for dynamic resource management in case of a failure, change in the configuration and initialization.

The aim of this thesis is to find ways for a distributed, self-configuring network architecture that offers a plug and play service to its devices. Therefore, we evaluated currently existing real-time networks and showed their shortcomings. Based on a literature study we took the three most promising candidates in the design space and extended them to fulfill requirements of a future avionic network. The proposed concepts were implemented in proof of concept demonstrators and backed by mathematical models. We evaluated the demonstrators during failure scenarios and showed the existence of dependability features needed for a future avionic network, like seamless fail-over. Building on this network, this thesis extended current self-configuration mechanisms to allow for plug and play networks with deterministic guarantees while enabling and making full use of the network's self-healing features. The mechanisms were implemented and tested with real-world scenarios. This thesis concludes with an outlook on open research points as well as missing items for the actual implementation in an aircraft.

Zusammenfassung

In vielen modernen Anwendungen werden heutzutage Echtzeitnetzwerke benötigt. Verschiedene existierende Lösungen wurden zu diesem Zwecke entworfen, jeweils spezialisiert auf branchenspezifische Anforderungen und Rahmenbedingungen. Um ein wirklich günstiges Echtzeitnetzwerk zu bekommen, wird im Moment ein gemeinsamer Standard entwickelt, federführend von der IEEE Arbeitsgruppe Time Sensitive Networking (TSN). Die Hauptmerkmale dieses neuen Standards sind unterbrechungsfreie Redundanz durch die Verwendung mehrerer disjunkter Pfade, niedrige Paketverzögerung für hoch-prioren Datenverkehr sowie verbesserte Synchronisationsmechanismen. Trotz diesen guten Eigenschaften, welche für Avionik-Zwecke noch zu evaluieren sind, wird der Standard nicht helfen, ein Netzwerk optimal im Hinblick auf dessen Dimensionierung und der Platzierung des Datenverkehrs zu designen.

Gleichzeitig ist in der Luftfahrtindustrie ein Trend zu mehr modularen und dezentralisierten Kabinengeräten zu sehen, welcher potentiell die Verfügbarkeit der Kabinenfunktionen durch höhere Fehlertoleranz und der Vermeidung der zentralen Server zur Gewichtsreduktion erhöht. Mit mehr Freiheitsgraden werden aber auch neue Mechanismen erforderlich, um den Einbauvorgang zu beschleunigen. Während Topologieerkennungs- sowie Adresskonfigurationsalgorithmen weit verbreitet sind, wurde wenig Augenmerk auf die speziellen Anforderungen im Avionik Bereich gelegt, wie zum Beispiel Echtzeitverhalten, Fehlerisolation und unterbrechungsfreie Redundanz. In einer solchen volldynamischen Kabine ist ein Cloud-ähnlicher Service vorstellbar, welcher dynamische Ressourcenverwaltung im Falle eines Fehler, einer Konfigurationsänderung oder während der Initialisierung vornimmt.

Das Ziel dieser Arbeit ist das Finden von Möglichkeiten zur Realisierung eines verteilten, selbstkonfigurierenden Netzwerks, welches diese Dienste als Plug & Play Verfahren für seine Geräte anbietet. Demzufolge wurden existierende Echtzeit-Netzwerke auf dessen

Schwächen untersucht. Basierend auf einer Literaturrecherche wurden die drei vielversprechendsten Kandidaten ausgewählt und um Konzepte zur Erfüllung der Anforderungen eines zukünftigen Avioniknetzwerkes erweitert. Die vorgeschlagenen Konzepte wurden prototypisch implementiert und durch mathematische Modelle abgesichert. Anschließend wurden die Prototypen im Normal- sowie Fehlerfall evaluiert und die Existenz und Funktion der Zuverlässigkeitsmechanismen gezeigt, wie beispielsweise unterbrechungsfreie Redundanz. Darauf basierend erweitert diese Thesis bestehende Selbstkonfigurations-Mechanismen um Plug & Play Funktionen mit deterministischen Garantien zu ermöglichen und dabei die vollen Möglichkeiten der Netzwerk-Selbstheilung auszuschöpfen. Die Mechanismen wurden implementiert und mit Daten realer Szenarien getestet. Die Thesis schließt mit einem Ausblick auf offenen Forschungsfragen ab und benennt fehlende Punkte, welche für eine echte Implementierung in Flugzeugen fehlen.

Contents

Chapter 1

Introduction

More and more industrial networks are seen in digital production floors and complex machines as part of digitization and data value generation trends. A lot of these networks also make a move away from smaller bus systems to open standardized unified communication networks that also offer real-time services. With the move to such converged networks and the increasing number of interworking systems, configuration effort of networks continuously increases and becomes more complex. To counteract the effects of this trend, this thesis presents a concept for a self-configuring real-time communication network with a special focus on avionics. The proposed future avionic network guarantees delivery of packets with no manual configuration and a seamless fail-over during faults.

For this network to be deployed in a critical infrastructure, the network needs to be dependable and able to deliver hard real-time guarantees. Further it should support a level of redundancy and self-healing that allows the network to deliver packets in the presence of failures. The traditional approach of using offline computed configuration files is to be replaced by a mechanism that allows all configuration to be computed and automated at run-time, either through a service discovery protocol or by a central database.

1.1 Context in Avionics

For a real-time network to be deployed in the avionic domain, multiple requirements have to be fulfilled which are explained in detail in Chapter 3.1. Its main characteristic has to be dependability with the ability to offer a selectable level of failure resistance from detection to masking. Further, such a network needs to offer hard quality of service

guarantees to its critical network services like flight control. It has to do so without a single point of failure and offer such guarantees according to a verifiable method that is eligible for certification. In order to be future proof, the network further has to support commercial off-the-shelf devices and integrate such in all self-configuration and traffic policing and shaping mechanisms.

1.2 Problem Statement

The research problem we want to answer in this thesis is divided into two major parts. An overview on why each of the research problems is a challenge is given in Chapter 3.2 and Chapter 3.3.

- **Research problem 1: Dependable networks that satisfy hard quality of service requirements and support seamless changes in its configuration at run-time.**

  A large amount of work exists on real-time networks that focuses on either hard quality of service guarantees, plug and play capabilities or ease of configuration. Combining multiple of these characteristics into one network is still an open problem under active research. Based on the networks found in literature we either extend currently existing networks to fulfill all requirements needed or propose a new solution that is based on open standards. The solution space is constrained by the special avionic requirements identified in Chapter 3.1 like needs for verification and based on commercial off-the-shelf devices.

- **Research problem 2: Means for assured self-configuration of networked devices with guaranteed properties and timing of the ensuing systems.**

  The future avionic network shall host a multitude of devices that can be easily attached and removed while the network takes care of guaranteeing dependability and timely configuration. Configuration protocols exist for address configuration and path reservation, however, mostly for closed and proprietary systems. A solution that works on standard open networks while guaranteeing dependability to the devices in terms of integrated seamless redundancy, multicast routing and automatic resource finding does not exist. Therefore, an easy to use plug and play mechanism needs to be developed that allows to handle the complexity of the real-time network.

The mentioned research problems will serve as a guideline throughout this thesis and act as a point of reference when interpreting the results achieved. In the conclusion we will elaborate how each of the problems was addressed and solved.

1.3 Contributions & Document Structure

This document is structured as follows. In Chapter 2 we present basic concepts and background in the areas of communication networks, certification in avionics and dependable systems. Further an overview of verification methods for real-time networks is given. In Chapter 3, an overview of related work in the fields of real-time networks and self-configuration methods is given. Each subchapter gives a state of the art overview of the respective subject and concludes with a table stating the challenges and research gaps in the respective area. The found gaps are then investigated and solved for real-time networks in Chapter 4 and for self-configuration in Chapter 5.

In Chapter 4 we develop in particular dependable networks that are configurable at runtime but still satisfy hard quality of service requirements. A high availability ring is extended to achieve deterministic behavior with minimal configuration effort. Next it is shown how to achieve guarantees in OpenFlow networks and a real-world aircraft switch configuration is loaded and used for evaluation against a state of the art flying avionic switch. Next the new upcoming networking standard called IEEE TSN [1] is evaluated with respect to aeronautical networks and a profile is proposed which selects a subset of the standard's wide variety of features to allow a potential use in avionics.

Afterwards in Chapter 5 we first give a short introduction to the current way of configuration of aircraft switches and show its shortcomings. Then two solutions are presented that allow for automatic configuration of the real-time networks found in the previous chapter. A plug and play deterministic mechanism is presented that allows devices to connect and disconnect seamlessly as well as offer self-healing features during faults. Further a mechanism is proposed how to safely move this configuration mechanism of the network into in-band channels.

Finally, Chapter 6 concludes this thesis by pointing to open problems and gives an outlook about possible future work in the area.

Chapter 2

Basic Concepts

The intention of this chapter is to give background information on the topics discussed within this thesis. Basic concepts are explained to keep explanations in later chapters concise. Further background information, e.g. on certification is given in order to understand why certain decisions were made within this thesis and why evaluations were done in a certain way.

Structure of this chapter

In Section 2.1 we give a definition of *dependability* and describe the attributes of a dependable system. Section 2.2 explains how *certification* in the aeronautic environment works with a special focus on hardware and software systems and introduces the most important analyses to be performed to obtain a certified system. In Section 2.3 we define *real-time systems and networks* and explain how real-time behavior can be achieved on a communication network. With real-time networks in place, Section 2.4 then explains how to *verify* and guarantee real-time behavior. Finally, Section 2.5 concludes the chapter and gives a *summary* on the topics presented.

2.1 Dependability

Dependability is defined as *the property of a computer system that reliance can justifiably be placed in the service it delivers* [2]. While each system will put a different emphasis on the properties of dependability, its main attributes are defined as availability, reliability, safety, confidentiality, integrity and maintainability. These characteristics are threatened by faults, errors and failures and can be averted by a combination of fault prevention, fault tolerance, fault removal and fault forecasting [2, 3]. The provision of integrity, availability and confidentiality attributes is denoted as security. Security, however, is not taken into closer consideration here due the segregation of networks in aircraft. Similarly we expect an industrial system to be maintainable and, therefore, neglect maintainability. All other attributes are explained and discussed in more detail in the following.

2.1.1 Availability

Availability (A) describes the *readiness for correct service* [2] of a system or in other words the fraction of time that a system is ready for usage. Such values will typically be measured for a set of similar devices resulting in values for the *mean time to failure* (MTTF) and the *mean time to repair* for a broken device (MTTR).

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \tag{2.1}$$

While the sum of MTTF and MTTR is known as *mean time between failures* (MTBF), in the industrial domain MTBF and MTTF are used interchangeably as $\text{MTTR} \ll \text{MTTF}$ if devices are assumed to be repairable. Typically, MTTF values are in the range of a hundred thousand hours. For example an avionic network switch has an MTTF of 100,000h [4]. Having the MTBF of a device, its probability to fail can then be calculated as follows.

$$\lambda = \frac{1}{\text{MTBF}} \tag{2.2}$$

Here λ denotes the likelihood of a device to fail. In aeronautics, typical values for λ range between 10^{-3} and 10^{-9} per hour and are further explained in Table 2.1. To increase the availability of a system, its sub-components can be arranged in parallel or duplicated in order to go from additive to multiplicative failure probabilities. For more background on availability we point to [5, page 136ff.].

2.1.2 Reliability

Reliability (R) is a measure of the *continuity of correct services* [2] or in other words the probability that a device will work for a stated period. Assuming a constant failure rate, the reliability can be calculated according to Equation (2.3) with t being the mission time.

$$R(t) = \exp(-\lambda\ t) \tag{2.3}$$

In reality, however, the simplification of using constant failure rates leads to inaccurate results. Instead of using a simple exponential reliability function, often Weibull, normal distributions or lognormal distributions are used depending on the type of device and measured the failures [6, page 32-28ff.].

$$F(t) = 1 - R(t) \tag{2.4}$$

The failure probability can then be calculated as seen above. For a critical aircraft component such as a network switch with a MTBF of 100,000h, such a value can then be calculated for a flight of 10 hours according to Equation (2.4). The probability of a single switch failure would therefore be F(10h) = 1 - $\exp(-10^{-5}\ 10h) \approx 0.01\%$. Most critical network traffic, however, is sent multiple times via redundant switches, therefore yielding even smaller probabilities. For more background on reliability we point to [5, page 128ff.].

2.1.3 Safety

Safety (S) is defined as the *absence of catastrophic consequences* [2] or the probability of the system staying in a safe state. Safety can be described as an extension of reliability, looking only at those reliability events that may cause safety hazards.

$$S(t) = \{s(t) \in (S_W \cup S_{FS})/s(\tau) \in (S_W \cup S_{FS}), \tau \in [0,t]\} \tag{2.5}$$

Mathematically S can be described as in Equation (2.5) with S being the possible system states where S_W represents operational states, S_{FS} safe failed states and S_{FU} unsafe failed states. For more background on safety we point to [7, 8].

2.1.4 Integrity

Integrity is the *absence of improper system state alterations* [2]. While it is important for the system to be up and running it is equally important that a system is not providing erroneous outputs that remain undetected.

Typical means to circumvent this from happening are built-in tests (BIT) that are executed during boot-up and run-time to check if the system is able to perform a set of functions correctly. BIT, however, is not capable to detect spontaneous computation errors. A second mean to ensure integrity is known as cross-monitoring, where two independent channels perform a computation and a cross-monitor compares the results by voting or by comparison.

Such techniques allow to have high integrity levels with wrong computations happening with a probability of less than 10^{-9} per flight hour. For more background on integrity we point to [5, page 139ff.].

2.2 Certification

With an increasing number of systems operating in critical parts of machines that can cause harm to people, governments and engineering societies have established guidelines and standards that regulate such critical parts and give common rules. Typical areas of certified products include aerospace, avionics, automotive, medical, nuclear, railway and others [9] and affected parts range from small parts to whole systems. Certification not only acts as a safety means for society, but also as a means for the vendor to limit responsibility for a system fault by having followed agreed rules.

In the aerospace domain almost all parts of a plane, including design, production and operation, are subject to certification and regulation. Applicable documents are called *Federal Aviation Regulations* (FAR), or *Joint Airworthiness Regulations* (JAR) for equivalent European versions, with different sections of it covering all parts of the aircraft. A manufacturer applying for certification has to demonstrate that applicable regulations are met for a certain product and, once successful, ends up with a *type certificate* (TC) for a certain aircraft type. For more background on type-certification we point to [6, page 9-1ff.].

Going down to the system level, ARP-4754 [10] classifies *failure conditions* (FC) that follow from potential faults of aircraft functions. Each function of the aircraft is mapped to one or several FC. Depending on the impact of the FC, a *Functional Development Assurance Level* (FDAL) is assigned to the function, e.g. any "catastrophic failure" condition to level A and "no safety impact" conditions to level E (see Table 2.1). *Design objective* (DO) documents describe these one-to-one mappings from failure classes to development assurance levels and impose analyses and probabilities to be reached for each FDAL class. For electronics and especially network devices in civil aviation FAR and JAR regulations typically point to two standards that are applicable:

1. *Software considerations in airborne systems and equipment certification* (DO-178C) from 2011 [11]
2. *Design Assurance Guidance for Airborne Electronic Hardware* (DO-254) from 2000 [12]

These two documents are guidelines on how to run a development process of a device and typically have to be used together for embedded hardware devices running a software as can be seen in Figure 2.1. DO-178 focuses on software developments with five major areas, (1) documentation integration and production, (2) system issues, (3) software development, (4) software verification and (5) software quality assurance. DO-254 has

three main areas including (1) hardware design life cycle management, (2) hardware design processes and (3) validation and verification processes. It especially concerns complex hardware like ASICs and FPGAs which have embedded firmwares that are not considered software and are therefore not covered by DO-178.

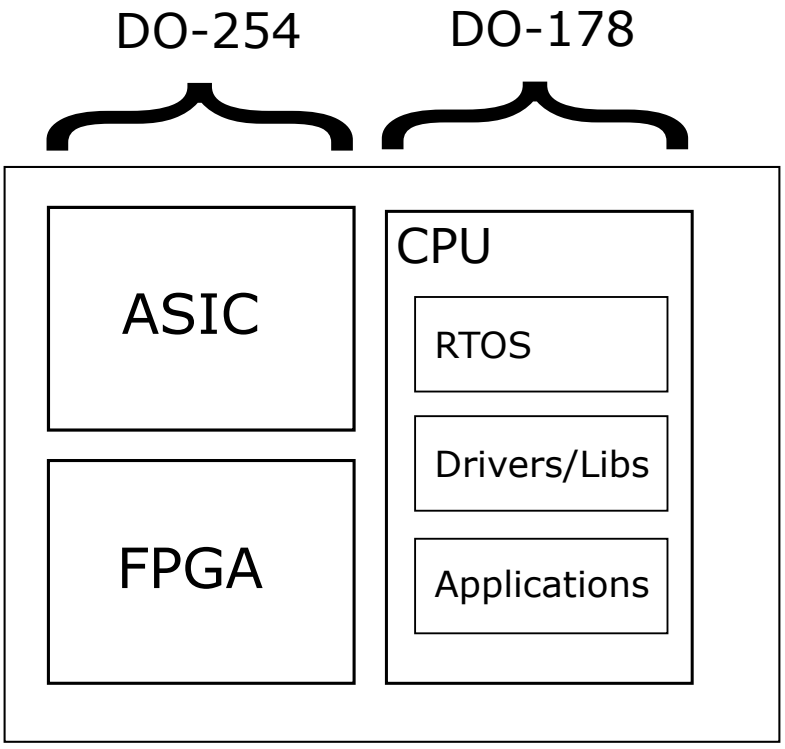


Figure 2.1: Overview of DO-178 and DO-254 applicability

2.2.1 Development Assurance Levels

Both DO-178 and DO-254 work with FDAL levels derived from the function's failure conditions. For each level different qualitative and quantitative analyses are defined by the DOs which have to be performed in order to show device maturity.

The *quantitative* part of device qualification is about how to reach its assigned failure probabilities in terms of availability (see Section 2.1.1) and reliability (see Section 2.1.2). Typical means for this are increasing MTBF of devices and/or installing multiple independent instances of a device in parallel. The second part of device qualification concerns *qualitative* design analysis that has to be performed in order to find for example common faults or implied failures through fault tree analysis. The most important quantitative and qualitative analyses are presented in the following.

2.2.2 Safety Analysis

To submit a certification request, each development assurance level asks for a set of *safety analyses*. Safety analyses can be performed in either a qualitative way, a quantitative way or a combination of both. Qualitative analyses contain the review of all factors that could have an influence on the safety of the system while quantitative analyses try to name the probability of such factors [13, Chapter 8].

For software and hardware devices DO-178 and DO-254 name a set of analyses that are listed in Table 2.1 and further elaborated in ARP-4761 [14]. While the list of analyses is lengthy, we explain here the most important ones to give an understanding of the topic.

2.2.2.1 Qualitative Design Analysis

Qualitative Design Analysis tries to find all parameters and describes what must go wrong in order to reach a safety critical state. The most common qualitative analysis is the *Common Cause Analysis* (CCA), which looks at common causes that have an impact on the independence of systems. CCA systematically tries to find such dependencies by deploying the following three sub-analyses ZSA, PRA and CMA [14].

Zonal Safety Analysis (ZSA)

Zonal Safety Analysis looks at the spatial distance of systems between each other. If two critical systems are placed too close to each other with one catching fire, the second system could be at risk and therefore is not seen as zonal independent anymore. In a typical aircraft environment with flight computers, this problem is solved by installing systems in different locations, e.g. one in the front and one in the back of the aircraft. For further details we point to [14].

Particular Risks Analysis (PRA)

Particular Risks Analysis looks at the risks that arise outside the system and can influence its operational capability. Typical examples named in ARP-4761 are fire, explosion of high energy devices like engine, fan or power generators, bird or lightning strikes. In difference to ZSA, PRA looks at all the zones at once while ZSA focuses on a single one. For each event that could cause a particular risk, a further study limiting this risk has to be performed. For further details we point to [14].

DAL Level	Severity	Probability per flight hour	DAL Definition	Typical Analysis	Exemplary System
A	Catastrophic	$< 10^{-9}$	Loss of aircraft unable to continue safe flight	Qualitative design analysis (PRA, ZSA, CMA) and quantitative design analysis (FTA, FMEA)	Air Data Computer, Fly by Wire, Cockpit Flight Display
B	Hazardous	$< 10^{-7}$	Large reduction in safety margin excessive workload for crew	Qualitative design analysis (PRA, ZSA, CMA) and quantitative design analysis (FTA, FMEA)	Instrument Landing System, Landing Gear Control
C	Major	$< 10^{-5}$	Physical distress to passengers Significant increase in workload	Depending on complexity; from qualitative design- to quantitative analysis	Navigation Systems, Yaw Damper, Auto-throttle Computer
D	Minor	$> 10^{-5}$	Slight reduction in safety margin Slight increase in crew workload	Design and installation approval to verify isolation from other systems	Air Conditioning, Cabin Lighting
E	No safety impact	N/A	No effect on safety, no effect on crew workload	Design and installation approval to verify isolation from other systems	In Flight Entertainment

Table 2.1: Development Assurance Levels (DAL) with failure probabilities and typical analyses to be performed (Compare [6, page 10-3ff.])

Common Mode Analysis (CMA)

Common Mode Analysis looks at the implementation of systems to verify that no dependence is found in the fault tree analysis. A typical example of a dependence is critical software installed on systems A and B with both systems made by the same manufacturer with the same software code and underlying computational engine. Due to their similarity, a common cause failure (e.g. a software bug) could affect the independence of the two. Therefore, for the previous example, such a critical system would have to be built by independent development teams with different software and computational engines in order to make the systems truly independent.

ARP-4761 in particular points here to hardware and software errors of any device which also includes a common time base for TDMA based systems. As such, if a time-synchronization based network is to be used in a critical system, it would have to be shown that time-synchronization failures are independent and do not propagate throughout the system. More on this can be found in Section 2.3.1.2. For further details on CMA we point to [14].

2.2.2.2 Quantitative Design Analysis

Quantitative Design Analysis puts numbers and probabilities on items found during qualitative design analysis and compares them to their development assurance levels as shown in Table 2.1. Probabilities are estimated with engineering methods, service history of similar devices or experience. For quantitative analysis typically two tools are used: (1) Fault Tree Analysis (FTA) and (2) Failure Mode & Effects Analysis (FMEA) with a relationship as shown in Figure 2.2. FTA typically starts from a top level event that causes a system failure and iterates down to single part failures. FMEA starts with part failures and tries to estimate the system failure probabilities.

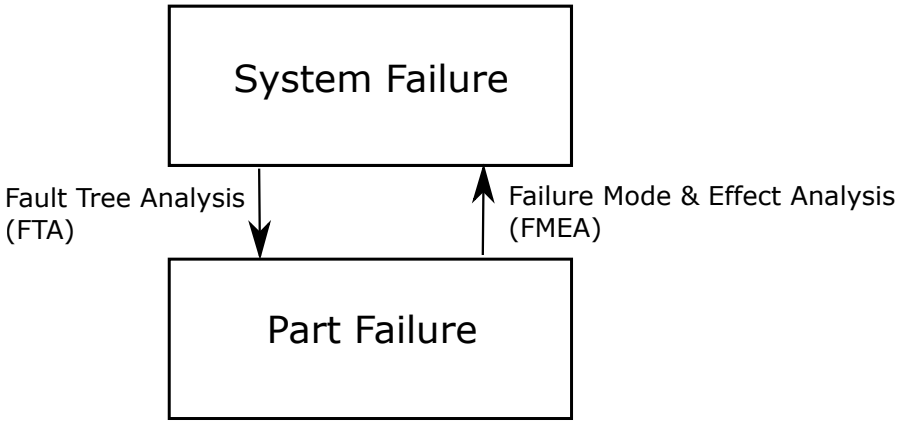


Figure 2.2: Relationship between FTA and FMEA

Fault Tree Analysis (FTA)

Fault Tree Analysis is a graphical method used to find single events that can lead to system level failures. It does so by dividing events into top-level events, intermediate events and basic events. Each event has a probability attached to it and by applying boolean algebra FTA tries to go from single events or a combination of basic events to a top level event. The top-level event has to be quite specific in order to keep the FTA concise. An example for a top-level event in an airplane is 'aircraft network down', with intermediate events being 'power cut' or 'switch loosing packets' and a single event being 'device stuck in transmission loop'. For further details we point to [14].

Failure Mode & Effects Analysis (FMEA)

Unlike FTA, *Failure Mode & Effects Analysis* uses a bottom-up approach to predict failures of the overall system. In FMEA, single events found in the FTA are iterated and the effect of this failure to the overall system is determined. Such an analysis has to be done for all sub-parts of a device, e.g. a complex digital device, including short and open circuits, inoperative modes, wrong calibration or loose contacts. For further details we point to [14, 15].

2.2.3 Mixed Criticality Certification

With evermore systems being introduced and the availability of more sophisticated and cheaper electronics the trend of integrating several systems across different safety levels into one machine is increasing in order to save weight and space. Unlike the common definition for mixed criticality of having high and low priority tasks where higher priority tasks suffer less or no performance degradation, we define mixed criticality in this thesis *as a system that runs multiple tasks, where none of the tasks can suffer performance degradation. A lower criticality task might need to be certified according to higher design levels or sufficiently separated by a mechanism that itself is of the higher design level.*

In avionics this concept is widely applied in *integrated modular avionic* (IMA) and was first introduced in Boeing's 777 in 1995 with its Airplane Information Management System (AIMS) [16]. While AIMS integrates multiple functions onto one system and deploys network communication between applications, this idea got later extended to sharing CPUs for multiple functions and having an open API to allow suppliers the development of such applications. Such applications are then run by a time and space partitioning hypervisor software that guarantees independence of its applications. This approach was finally allowed for certification in DO-297 [17].

The way DO-297 allows mixed criticality is by using hardware certified according to its highest DAL level and by adding a space and time partitioning hypervisor with a standard API to allow computation as well as communication (e.g. ARINC 653 [18]), which can also be used for inter-partition communication.

For communication between different modules, separation has to be ensured by the network. This is typically done with virtual links as in ARINC 664 which will be later explained in Chapter 3.2.1.1.

In order for an IMA system to be certified, a maximum worst-case execution time of each task has to be determined. While this is supported by the state-of-the-art for single core processors, this poses still some challenges for multi-core processors due to the sharing of the same memory and is still under active research [19–21].

Second generation IMA systems take this approach a step further and remove the physical mapping of tasks onto specific hardware but allow distributing the functions over all available resources in the network with real-time communication links in between. This approach is generally known as *distributed integrated modular avionic* (DIMA) with active research on how to find an optimal mapping of functions to devices [22, 23].

2.2.4 Commercial off-the-shelf Devices

> **Note** This section mentions EASA's certification memorandum SWCEH-001 [24]. This document is not officially accepted by Airbus as of today and certification is done in a similar, undisclosed way.

A key enabler for many modern systems like fly-by-wire are *commercial off-the-shelf* (COTS) devices like processors and highly integrated system-on-chips that offer enough computational performance for demanding tasks at affordable cost. The need to allow COTS devices in certified systems has been recognized by certification authorities and was captured in Section 11.2 and 11.3 of DO-254 [12] as well as in a certification memorandum [24].

These two documents give guidelines on how to allow COTS devices in all DAL levels. The main idea is that criticality levels are assigned per function and not to individual devices. Therefore, COTS devices can be integrated in the overall design process if maturity is ensured. The process of showing this maturity for COTS devices is done by collecting so called *certification credit*.

Certification credit can be obtained by showing that a device is undergoing quality control at the manufacturer including change management processes and that the device complies

with needed operational constrains like temperature, vibration, operating voltage and radiation. A device's service record of operation in other critical systems is also taken into consideration. For the most critical development levels a service record of more than 10^6 hours in previous aircraft and/or safety applications in space, airborne military, nuclear, medical or railway domains has to be shown [24].

When enough certification credit has been obtained, it furthermore has to be made sure that a COTS device cannot create a *stand-alone catastrophic* event. This single point of failure is assumed to happen through anomalous behavior found in identical COTS devices and therefore has to be averted by e.g. using multiple COTS vendors running independent software programs. Such considerations have to be found and documented during CMA analysis. Another requirement for COTS devices is that their run-time behavior has to be deterministic [24]. While this is state-of-the-art for single core processors, this proves to be problematic for multi-core processors. See Section 2.2.3 for more details on multi-core WCET.

2.3 Real-Time Communication Systems

While there are multiple notions of real-time systems, in this thesis we use the following definition.

"A real-time computer system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced" - Kopetz [25]

While the above definition talks about the whole system, on a task level it can also be defined as a system that has the objective to meet the individual timing requirements of each task. Depending on the deadline and usability of the task's result real-time systems can further be categorized in the following categories [26]:

1. **Hard** real-time system: A system or task is considered hard real-time, if producing the result after its deadline may cause catastrophic consequences on the system under control [26].
2. **Firm** real-time system: A system or task is considered a firm real-time task, if producing the result after its deadline is useless for the system, however, does not cause damage [26].
3. **Soft** real-time system: A system or task is considered a soft real-time task, if producing the result after its deadline has still some utility for the system, although causing performance degradation [26].
4. **Best-Effort** system: A best-effort system or task does not have any requirements regarding its deadline [26].

The same definition can be applied to real-time communication networks, if a task's deadline is considered the deadline until successful delivery of information to the receiving node. In a network this is represented by the end-to-end latency of a packet. Examples for hard real-time latencies in a network typically include low-level control loops or critical sensory data; firm real-time latencies include audio/video transmission and on-line image processing while soft real-time latencies are typically used for saving report data or some paying customers.

The mechanisms to achieve bounded end-to-end latency in a network are presented in the following and are agnostic to the technology used. An overview with concrete technologies applying these mechanisms, e.g. Ethernet, is given in Chapter 3.2.

2.3.1 Mechanisms for Bounded Latency

To enable real-time access and communication on a shared network deterministic access to the bus is needed. The literature [27] distinguishes between the following categories, as can be seen in Figure 2.3, which are all types of probabilistic or deterministic medium access mechanisms.

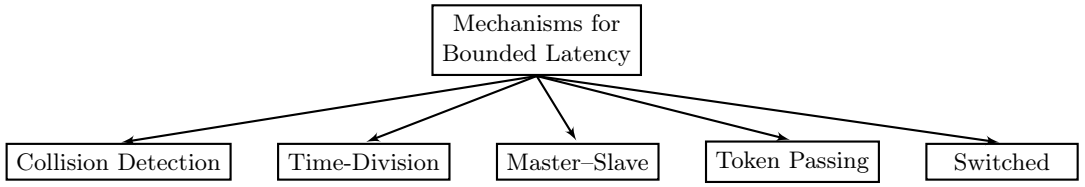


Figure 2.3: Overview of Mechanisms for Bounded Latency

2.3.1.1 Collision Detection

Collision Detection aims to provide probabilistic guarantees for the medium access by detecting collisions on the bus followed by random wait times before a retransmission. While this does not guarantee a deterministic access, a high probability can be obtained when considering low traffic networks in terms of link utilization and message length. Schneider [28] demonstrated that a message loss probability of less than 10^{-7} can be reached in classic Ethernet when allowing for hundredfold over-provisioning of the actual bandwidth needs.

To increase utilization, Le Lann and Rivierre [29] propose a method that removes the randomness of the retransmission times but replaces it by a structured value found through the use of a structured binary tree. Lower priority sources will stop sending in favor of higher priority sources, thereby limiting the number of steps until the current highest priority is able to transmit successfully. Despite its bounded access time this approach will deliver very pessimistic values in a worst-case analysis with a larger number of stations and priorities.

2.3.1.2 Time-Based Mechanisms

Time-Division Multiple Access (TDMA) mechanisms are based on a global time with slots assigned to each station during which such a station is allowed to send. In simple systems such slots are static, however, slots can also be assigned dynamically to each node in each TDMA round with different duration like for example done in Lee [30].

For TDMA to work, time has to be synchronized across the nodes. This is typically done in a distributed fashion [31] or through a special station sending time beacons for the nodes to synchronize [32]. The problem that arises from synchronization, regardless of the level of redundancy of the system, is that clock synchronization inevitably becomes the bottle-neck and will impose a system failure when clocks become deskewed. Provably correct fault-tolerant clock synchronization algorithms are presented in [33], which are all based on three or more global clocks. Due to the complexity of these mechanisms, as of now, none of them have been deployed in civil avionics yet due to certification reasons [5].

2.3.1.3 Token-Based & Master-Slave Mechanisms

In *token-based* mechanisms stations are only allowed to send when owning the token. The token is a special packet passed around in the network from one station to the next or from a master to the slave. The time the token resides within each station has to be bounded in order to achieve deterministic behavior.

The major drawback of this mechanism is the possibility of loosing a token. This can be circumvented by triggering a periodic rotation of the token or by recreating it during loss like proposed in [34], which then falls back to a TDMA like approach.

Similarly, in a master-slave scenario stations are only allowed to send upon request by a master. Typically, such a configuration leads to high under-utilization of the network as each communication has to be registered first with the master.

All mechanisms presented so far, however, do not perform well in case of a failure. If a device crashes and gets stuck in transmit mode, it will effectively block the bus and make it unusable for any other device. Such a failure is also known as a *babbling idiot* in the literature.

2.3.1.4 Switched Mechanisms

To circumvent a babbling idiot device, switches have to separate the network into point-to-point links and provide private collision domains. Medium access is therefore deterministic. However, switches have to buffer arriving packets in queues at their output ports. These queues may build up and drop data if too many packets arrive within a short interval. Different methods similar to medium access control can be applied in order to limit the intake of the switch. Such network technologies based on switching are presented in Chapter 3.2.

2.3.2 Heterogeneous Latency Requirements

In the aeronautical environment networks are typically divided by criticality into separate networks with each network having a different set of requirements concerning their real-time behavior. Usually the networks are split into 4 domains, as defined by ARINC 664 Part 5 [35]. However, networks can sometimes be combined to one heterogeneous multi-domain network.

Aircraft Control Domain (ACD)
The *aircraft control domain*'s primary function is to enable the safe operation of the aircraft. It is characterized by its slow rate of change as well as high demands for real-time behavior in terms of latency and jitter on the network level. Nearly all devices will be subject to certification in this domain and are therefore rather expensive. Typical systems are flight control, cabin control and data loading [35].

Airline Information Services Domains (AISD)
The *airline information services domain* supports the operation of the aircraft by an airline by hosting administrative services and offering inter-connectivity between separated domains, such as avionics to in-flight entertainment gateways. Like in ACD, AISD systems should protect themselves from lower criticality systems. Their life-span is shorter than those of ACD systems due to faster upgrade cycles in this domain. Typical systems in this domain include cabin management systems like light controls and airline business functions [35].

Passenger Information and Entertainment Services Domain (PIESD)
The main function of the *passenger information and entertainment services domain* is to provide entertainment services to passengers such as the in-flight entertainment system. The typical life-span of these systems is quite short due to the multimedia nature of devices. The network itself has to offer some soft real-time guarantees in order to guarantee high quality video experiences.

Passenger Owned Devices Domain (PODD)
The *passenger owned devices domain* contains any device brought on-board by a passenger that is connected to the aircraft either through a wireless or wired connection. Service in this network will typically be provided by PIESD services. The PODD network itself is considered as highly dangerous to the safety of the aircraft and is therefore not connected to any of the higher criticality domains.

Depending on the domain and criticality of a system different real-time behaviors are expected as summarized in Table 2.2. Some networks have to offer a multitude of hard, firm and soft real-time guarantees.

Domain	**Application**	**Real-Time Behaviour**	**Delay Requirements**	**Jitter Requirements**
Aircraft Control Domain (ACD)	Flight control	Hard	10−100ms	10−100ms
	Cabin control	Firm	10−100ms	10−100ms
	Data loading	Soft	100ms−1s	100ms−1s
	Interactive audio	Soft	10ms	10ms
Airline Information Services Domains (AISD)	Cabin management	Firm	100ms	100ms
	Business functions	Soft	100ms	100ms
Passenger Information and Entertainment Services Domain (PIESD)	Flight information	Soft	100ms	100ms
	Audio streaming	Soft	10ms	10ms
	Video streaming	Soft	10ms	10ms
Passenger Owned Devices Domain (PODD)	Video streaming	Soft	100ms	100ms
	Internet access	Best-Effort	$\geq$ 100ms	$\geq$100ms

Table 2.2: Latency and Jitter requirements of aircraft functions with their respective domains (Table adapted from Geyer [36])

2.4 Verification of Performance Bounds

Certification demands that deadlines of real-time systems are provably met. Several ways have been proposed to calculate upper bounds in networks for latency, jitter and buffers, which can traditionally be divided into (a) analytical methods and (b) measurement based methods.

Only analytical methods give strong mathematical upper bounds that can be proven, however, usually at the cost of being too pessimistic due to simplifications for computability. The task of finding tight bounds has been proven to be NP-hard [37] for the general case. Typical analytical analyses are *network calculus*, *model checking* and *trajectory approach* which will be described in the following.

Network measurements on the other hand reveal more realistic average results. However, long runs with all devices are needed to actually observe the worst-case if it is found at all. A complex scenario may be too complicated to be replicated in a lab set-up and its bounds are therefore difficult to find and prove with measurements.

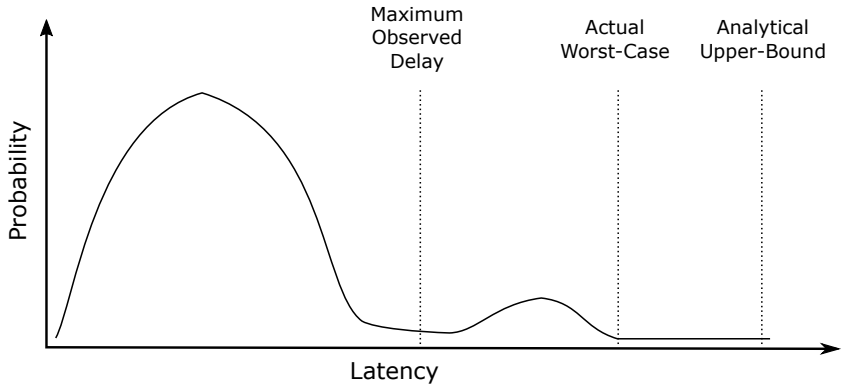


Figure 2.4: Observed latency vs. analytically calculated bound
(Adapted from [36])

An intermediate between the two are *hybrid* solutions that are based on probabilities. Typically hybrid solutions use *network simulations* which try to find worst cases by the use of *monte carlo methods* and assumptions. These methods model the network and assume the traffic characteristics to be part of a particular probability distribution. To a certain probability the actual worst-case will be found which is then combined with analytical methods to give a probabilistic worst case (PWCET) [38], as shown in Figure 2.4.

2.4.1 Network Calculus

The main method used for verification accepted by aviation authorities for certification is *network calculus* (NC). Network calculus was first proposed by Cruz [39] and later extended by Le Boudec and Thiran [40] to so called Min-Plus algebra. NC is an algebraic framework that models the characteristics and behavior of traffic in the network.

All data flows R in a network have to be mathematically described in so called arrival curves. An arrival curve $\alpha(t)$ describes the number of bits seen per time of one data flow in the interval [0,t] and is non-decreasing with time t. An example of a simple arrival curve is $\alpha(t) = rt$. $R(t)$ is said to be limited by $\alpha(t)$ if for all $s \leq t$:

$$R(t) - R(s) \leq \alpha(t - s) \tag{2.6}$$

When the data flow R(t) is passing through a system $\mathcal{S}$ resulting in an output flow $R^*(t)$, the system is said to offer a service to its flows if $\forall t \geq 0 \exists t_0 \geq 0$ with $t_0 \leq t$ such that:

$$R^*(t) - R(t_0) \geq \beta(t - t_0) \tag{2.7}$$

A typical service curve is $\beta(t) = R\lceil t - T \rceil^+$ with R being the bandwidth and T the static switching delay. Having the input and output function of a constrained flow α through a system offering a service β, the needed buffer space B for all t can then be calculated by:

$$B = R(t) - R^*(t) = \sup_{s \geq 0} \{\alpha(s) - \beta(s)\} \tag{2.8}$$

Similarly, its delay bound D can be calculated too:

$$D = \sup_t \left\langle \inf_{\tau \geq 0} \{\alpha(t) \leq \beta(t + \tau)\} \right\rangle \tag{2.9}$$

Further functions are defined to calculate the output function $R^*(t)$ of a system by using the service and arrival curves of all flows, as well as simplifications that take bursts into consideration. For more background on network calculus we refer to [40–42]. Also we point the interested reader to Frances et al. [43] on how to further optimize network calculus results by using priorities.

Example Avionic Calculation

We show here a example calculation of buffer and delay times for one flow in a typical avionic network using *Avionics Full-Duplex Switched Ethernet* (AFDX). The flow f_0 is assumed to be of a maximum packet size of 128 bytes transmitting every 2ms. It can be modeled with an arrival curve using a token-bucket model $\alpha(t) = (rt + b)$ for $t > 0$ with $r = \frac{\text{maximum packet size}}{\text{interval time}}$ and b equal to the maximum packet size, resulting in $\alpha_0(t) = \frac{128*8\text{bit}}{0.002\text{s}}t + 128*8\text{bit}$. The switch $\mathcal{S}0$ in the networks offers 100 Mbps line rate with a static forwarding delay of $20\mu s$, therefore giving $\beta_0(t) = 100000000\frac{\text{bit}}{s}\lceil t - 0.00002s\rceil^+$.

The path of the flow will look as follows: Source $f_0 \longrightarrow$ **Switch** $\mathcal{S}0 \longrightarrow$ Sink f_0

The buffer needed in the Switch $\mathcal{S}0$ to not run into any overflow can now be calculated according to Equation (2.8):

$$\begin{aligned} B^{f_0} &= \sup_{t\geq 0}\{\alpha_0(t) - \beta_0(t)\} \\ &= \sup_{t\geq 0}\{\frac{128*8\text{bit}}{0.002\text{s}}t + 128*8\text{bit} - 100000000\frac{\text{bit}}{s}\lceil t - 0.000020s\rceil^+\} \\ &= \frac{128*8\text{bit}}{0.002\text{s}} * 0.000020s + 128*8\text{bit} = 1034\text{bit} \end{aligned}$$

Similarly the queuing latency D can be calculated according to Equation (2.9):

$$\begin{aligned} D^{f_0} &= \sup_t \left\langle \inf_{\tau\geq 0}\{\alpha_0(t) \leq \beta_0(t+\tau)\} \right\rangle \\ &= \sup_t \left\langle \inf_{\tau\geq 0}\{\frac{128*8\text{bit}}{0.002\text{s}}t + 128*8\text{bit} \leq 100000000\frac{\text{bit}}{s}\lceil t + \tau - 0.000020s\rceil^+\} \right\rangle \\ &= \sup_t \left\langle 100000000\frac{\text{bit}}{s}\lceil t - 0.000020s\rceil^+ = 128*8\text{bit} \right\rangle \\ &= 0.00003024s = 30.24\mu s \end{aligned}$$

For an automated calculation of this we refer to a tool called *DiscoDNC* by Bondorf and Schmitt [44].

2.4.2 Model Checking with Timed-Automata

Another way to model timing properties of real-time systems are timed automata as proposed by Alur and Dill [45]. A timed automata is a finite automaton that has its state-transitions constrained by time. Such automata can then be checked for safety and liveness properties with formal methods.

Charara et al. [46] adapted this approach to AFDX by making use of the periodicity of virtual links used in avionics. It was shown, that timed-automata give exact worst case latencies for small networks, however, lack scalability due to the large state space in realistically sized networks.

2.4.3 Trajectory Approach

While network calculus focuses on the maximum delay introduced on every hop, the trajectory approach [47] looks at the path a flow is taking through the network. The analysis is based on busy periods, similar to WCET computation in process scheduling.

Bauer et al. [48] adapted this to AFDX and showed, that computed results in the trajectory approach were in some cases better than those of network calculus, however, did not result in provable bounds in all cases.

2.5 Summary

This chapter gave background information in order to understand this thesis. Basic terms needed for dependability, like availability and reliability, were defined and discussed. It was shown how to calculate values like mean failure times which are needed for certification. An overview of the most important analyses in avionic certification processes was given and explained.

Afterwards real-time systems and networks were introduced and methods on how to validate and verify their upper bounds. It was discussed that, while other methods do exists, network calculus still is the favored method used by certification authorities and is therefore used in the following work within this thesis.

Chapter 3

Related Work

The goal of this chapter is to give an overview of related work on the fields of real-time networks as well as self-configuration. Requirements and challenges specific to aeronautics are introduced and derived into categories for related work to be evaluated. Further research gaps in literature are shown and will be focused on in the following chapters of this thesis.

Structure of this chapter

In Section 3.1 we give an overview of the design requirements expected from a future avionic network and the challenges specific to the aeronautical industry. Also we introduce the concept of fault tolerance. We then give an overview of related work in Section 3.2 with a focus on real-time networks and highlight current challenges and gaps in literature with a tabular overview. In Section 3.3 we give a state of the art overview on self-configuration protocols and mechanisms for assured self-configuration with guaranteed dependability. We conclude with a table giving an overview of challenges and gaps for self-configuration mechanisms.

3.1 Requirements and Challenges

This section describes the design goals for a future avionic network in terms of real-time and fault behavior as well as the need of self-configuration. We set a particular focus on having a dependable system, as faults of the system would lead to an increase in crew workload or worse and therefore have to be avoided. Due to the shape of the aircraft, line topologies are common in current systems. However, they are steadily being replaced with ring or meshed topologies to offer better fault masking and reliability. Requirements presented in this section will be used in the discussion of related work and summarized in Tables 3.2 and 3.3.

3.1.1 Real-Time

Here we list requirements the network needs to fulfill in terms of real-time mechanisms in order to be considered for a future avionic network.

Hard quality of service guarantees

The network needs to be able to implement hard quality of service guarantees. Such guarantees have to be provable through formal verification as for example those presented in Chapter 2.4. Also, the same mechanism needs to ensure that ill-behaved end-systems will not interfere with well-behaved end-systems.

Certifiable

While formal verification proofs system guarantees as long as the system is in a defined state, it does not consider functional requirements that could alter the system state or an abstracted model that inaccurately models the reality. As explained in Chapter 2.2, safety certification demands to avoid single points of failures and common mechanisms. As such, time synchronized systems have not acquired the highest level of safety certifications yet. We point the interested reader to Chapter 2.2.2 for more details.

COTS support

The network shall support devices that are not aware of the quality of service mechanisms. Devices shall be able to send at any time using standard Ethernet with the

network's job being enforcement of compliance to any predefined access profile and traffic characteristics. A typical example of this is a commercially available wireless access point to provide cabin functions to crew members.

3.1.2 Fault Tolerance

We define here the terms to classify and evaluate the related work based on fault tolerance. Background on dependable systems was already given in Chapter 2.1.1. Networks can be distinguished with respect to their ability to handle faults. We separate the networks here based on fault detection, fault isolation and fault masking.

Fault Detection

The first step in fault handling is to detect a fault. Different mechanisms can be applied specifically for networks. One is Ethernet's cyclic redundancy check (CRC) that detects bit-flips in messages that can happen due to cosmic radiation at high altitudes. This is also known in literature as a single upset event [49].

Fault Isolation

Once a fault is detected, the next step is to isolate the fault in order to prevent the fault to spread to other parts of the system. In terms of networks, this can be done by dropping frames outside their expected traffic envelope or by dropping frames with invalid CRC checksums.

Fault Masking

An alternative to isolation is fault masking, which is usually done by adding additional information. This can be done per message by introducing extra bits that allow for forward error correction (FEC), or per stream by duplicating all messages with duplicate delivery across disjunction paths [2, see A.2].

3.1.2.1 Fault Assumptions

In order to design fault tolerance mechanisms, we introduce the following fault assumptions. We follow the failure modes according to IEC 61508-2 [50] later refined by Abuteir et al. [51]. Through independent design and fault containment mechanisms, we assume

each node, switch and physical link to be a *fault containment region* (FCR) that keeps operating independently despite possible faults outside the FCR. The failure modes that can occur within such a FCR are as follows:

- *Component Crash*: Physical fault of the node that results in a permanent fault and no output being generated.
- *Link failure*: The permanent or sporadic failure of a physical link between two nodes or switches.
- *Omission*: A sender is not able to generate or forward a message which results in frames being dropped.
- *Corruption*: Transmitted data might get changed during transmission either through a hardware fault or EMI influences.
- *Delay*: Messages might get delayed across the transmission path.
- *Babbling idiot*: A node or a switch generates messages at wrong points in time or with wrong frequency.
- *Masquerading*: Nodes assume the identity of another node B and possibly use up B's reserved traffic resources or MAC address in best-effort communication.

We assume the occurrence of a failure in one FCR in the system which is usually denoted as a *single fault hypothesis*.

3.1.3 Self-Configuration

To reduce the overall workload and sources of failures, the network should also be self-configuring according to the following requirements.

Automated configuration at run-time

Newly connected devices are to be automatically configured without manual intervention. This includes a service discovery as well as reservation process which ends with a guarantee given by the network towards its flows. If there are insufficient resources or no configuration can be found, the end-system has to be informed.

Centralized In-Line Configuration

Due to complexity of the system and certification reasons, a central entity will need to supervise all configuration activity. This central entity shall not infer extra cabling or installation effort and therefore demands in-band configuration channels that run through the operational data network.

Commercial off-the-shelf (COTS) support

COTS devices not aware of any service discovery and registration mechanisms shall be usable in the network. A central entity can detect these devices and generate access rules based on characteristic features, e.g. the device's MAC address.

3.2 Aeronautical Networks

This section gives an overview over communication networks used in avionics and surrounding fields. A brief historical overview of the development is given and the main networks are then explained in detail. For each network related work is referenced and summarized.

Starting off in the 50s and 60s, all aircraft were built using analog signals for their electrical systems. From around the 70s, the first digital computers with analog-to-digital converters were introduced allowing new functions, mostly using point-to-point and parallel star topologies with unidirectional communication. This, however, did not scale well and the need for digital bus systems arose [52].

In 1973 the first serial bus was standardized with the name of MIL-STD-1553. The standard got updated to 1553A and later 1553B, which is still in use nowadays in military planes. Although civil aviation groups were interested in the MIL bus, it was too costly nor fitting civil certification authorities' demands and they started their own related standard ARINC 419. This bus later got updated to ARINC 429, which remained a widely used standard till the 1980s [52].

With more digital systems appearing, the old buses got improved in terms of speed and flexibility into ARINC 629 in the 1990s. Together with CAN, a digital bus system developed for low cost automotive equipment, it is still today the main bus system connecting low-power edge-devices to a fast communication backbone. Since the A-380, standard Ethernet, with some extensions called ARINC 664, has been the backbone communication network for all modern civil aircraft. While ARINC 664 offers faster speed, its static configuration makes it hard to configure during aircraft change or changing traffic demands.

An overview over aeronautical networks currently being operated in civil aircraft can be seen in Table 3.1. For more information on the history of aeronautical networks we refer the interested reader to Munoz-Castaner et al. [52].

Types of networks

Besides the different network technologies, also different types of networks with independent use-cases are running side by side in a typical civil aircraft. These are typically distinguished based on their function's assurance level (see Chapter 2.2) and the network's lifetime.

1. **Avionic networks** are used for critical control functions of the aircraft with static configuration, hard guarantees for each communication channel, zero packet loss

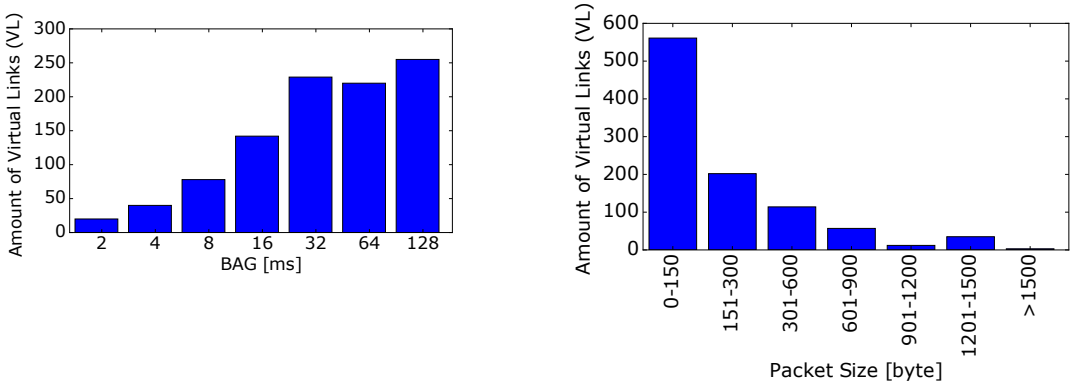
Year	Standard	Section	Topology	Aircraft Models
1973	MIL-STD-1553	3.2.1.3	Bus	F-16, F-18
1977	ARINC 429	3.2.1.3	Bus	Airbus A-330
1985	CAN	n/a	Bus	Airbus A-380, Boeing B-787
1989	ARINC 629	3.2.1.3	Bus	Boeing B-777
1998	TTP	3.2.1.2	Star	Airbus A-380, Boeing B-787
1999	ARINC 664 (AFDX)	3.2.1.1	Tree	Airbus A-380, Boeing B-787

Table 3.1: Summary of Aeronautical Networks (Compare [5, 52])

and designed to the highest assurance levels. The typical lifetime of such a network is typically the whole aircraft life, e.g. 20 years and more. Due to certification requirements it is hard to change any configuration in this network after initial deployment. An exemplary network load of an avionic network in an Airbus A-380 aircraft can be seen in Figure 3.1.

2. **Cabin networks**: Typical functions of the cabin network include services like oxygen control and safety audio announcements. While these services need guarantees, also video traffic and other best effort passenger traffic with heterogeneous time requirements can be found in this network. The lifetime of such a network is typically till a major cabin retrofit, around 10 years or more, and offers higher bandwidth than an avionic network.

3. **Entertainment networks** are a special kind of network only for entertainment purposes like movies and other in-flight entertainment services. While such networks do not pose safety criticality, non availability of it could result in loss of business for the airline due to non-existent entertainment functions. The typical



(a) Time between two consecutive packets (BAG) of the same connection

(b) Packet Size

Figure 3.1: Overview of traffic in an A-380 avionic network with virtual links (Compare [46])

lifetime of an entertainment network is usually till the next entertainment upgrade, around every two years. Such networks typically consist of fast fiber-optic backbones and are fully in the hand of the airline.

3.2.1 Strictly Aeronautical Networks

Deterministic networks are widely available in literature and practice. This section gives an state-of-the-art overview of deterministic networks used currently in avionics with a focus on mixed-criticality support, fault behavior as well as reconfiguration. Furthermore, this section presents open research questions as well as related work in the area of networks.

3.2.1.1 ARINC 664 P7 Avionics Full-Duplex Switched Ethernet

The main technology used in aeronautics to build safety critical networks is *Avionics Full Duplex Ethernet* (AFDX) [53] as defined in the ARINC standard 664 part 7. AFDX is based on Ethernet and offers real time guarantees through statistical multiplexing without the need for time synchronization. All packets are mapped to a flow called the virtual link (VL). Virtual links are unidirectional flows from one end-system to one or more end-systems. Identification of VLs is done via the packet's destination MAC address as shown in Figure 3.2. The upper 32 bits are constant while the lower 16 bits represent the virtual link. Each VL is characterized by a *Bandwidth Allocation Gap* (BAG), the time between two consecutive packets of the same flow, and a maximum (s_{max}) and minimum (s_{min}) frame size. Switches enforce compliance to this traffic description and can thus offer a guaranteed service time. Network calculus is then used to determine the worst-case latency as well as switch buffer sizes through the whole network. Network calculus is explained in detail in Chapter 2.4.

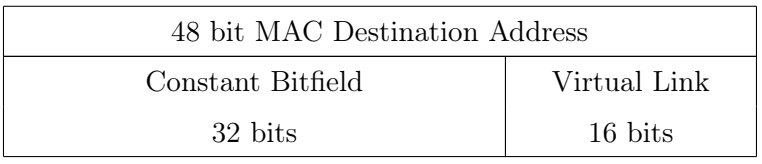


Figure 3.2: AFDX Destination MAC Structure [53]

While this mechanism works for asynchronous real-time traffic, it has not been specified for mixed criticality traffic, however, it would be theoretically possible. Failure detection happens in multiple ways: Firstly Ethernet's CRC checksum checks for bit errors at every hop along the way. Secondly a check for compliance to predefined traffic characteristics

is performed. As soon as any of the two faults is detected, the packets are dropped, therefore not influencing functionality of other services or the network itself. In AFDX no reconfiguration is supported in any way. A configuration is calculated off-line, verified and uploaded into the switches. A change in configuration usually does not take place.

With the presented limitations, scientists aimed to optimize AFDX in mainly two areas: (1) the descriptions of upper bounds and (2) the placement of flows in the network.

Le Boudec and Thiran [40] described a way to model network traffic and to calculate upper bounds known as network calculus, which is the main method that is used for certification today. As explained in Chapter 2.4, these bounds were found to be very pessimistic and have been subject to further research to bring theoretical and practical measurement results closer together.

Multiple approaches have been studied to optimize this, e.g. making use of specific effects of Ethernet [54] or a combination of network calculus and scheduling theory [55]. Further, due to the nature of certification requirements which allows for a probabilistic bound of network functionality, approaches using stochastic models have been used [56]. Based on stochastical models, Geyer et al. [57] proposed a way to extend this work to be used on stateful TCP connections.

The second area of research in AFDX is the placement of flows in the network across a multitude of possible routes. Heidinger and Kammenhuber [58] used mixed-integer linear programming to optimize flow placement in a network. Sheikh et al. [59] discussed, how to model individual links, in terms of having multiple small sized or fewer, larger sized, virtual links. Similar Cha et al. [60] looked on how to group multiple virtual links together into several larger ones.

3.2.1.2 SAE AS6003 Time Triggered Protocol

Time-triggered protocol (TTP) is an open protocol for bus and star topologies working as a masterless synchronized system. TTP is based on work from Kopetz and Grünsteidl [61], which got standardized into SAE AS6003. It is a TDMA based system with same length rounds, where each round is divided into slots with dynamic length for the nodes to send. The physical layer itself is not part of the specification. However, commercially available nodes use RS-485 as a physical layer with up to 5 Mbps asynchronous data rate and 25 Mbps synchronous data rate [62]. Each node is connected to two separate channels as shown in Figure 3.3.

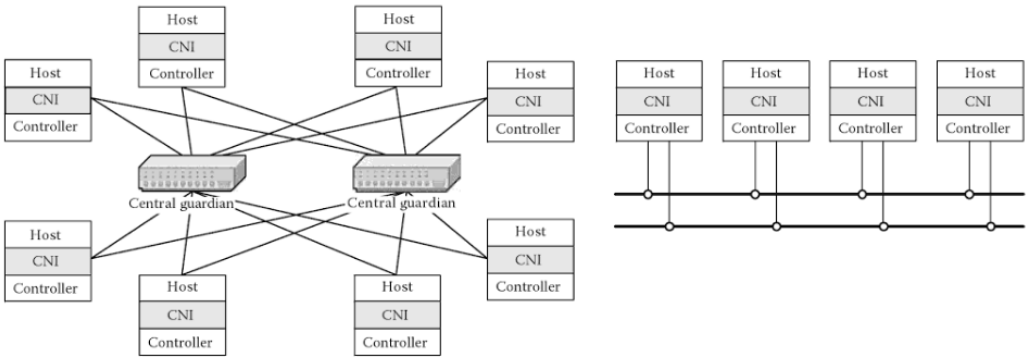


Figure 3.3: TTP Star vs Bus Topology (via [62])

While all slots of the communication schedule are preplanned, remaining slots can be divided between the nodes to allow for asynchronous events or mixed-criticality communication, therefore offering some flexibility if remaining slots are preallocated during design time [63]. Besides a CRC checksum, every node contains a transaction table of all slots used in the network. By listening to the bus, each node can therefore verify its neighbors and itself. A bus guard in each node will disconnect the local node once the local transaction table does not match to what is seen on the network.

The local bus guard is independent of the sending functions of each node. In a star topology, this guardian is placed in the central switch. Multiple works on the topics were presented for cases where time clusters can exist in parallel [61] and solutions to achieve fail-silent behavior in such cases [64].

An extension of TTP is *Time-Triggered Ethernet* (TTEthernet) [65] as standardized in SAE AS6802, which offers TTP like functionality based on Ethernet. TTEthernet offers support for time-triggered, rate-constrained and best-effort messages. Rate-constraint traffic works similar to AFDX, whereas the time-triggered messages work on a global TDMA like schedule. Abuteir et al. [51] proposed a TTEthernet based ring solution with selective fault-tolerance and support for mixed-criticality. Lauer et al. [66] discuss its use on aircraft.

3.2.1.3 Legacy Networks

The next section gives a brief overview of legacy networks that used to be deployed in aircraft, however, have mostly been replaced by AFDX and TTP.

MIL-STD-1553B Data Bus

MIL-STD-1553B is a military bidirectional linear data bus with a centralized controller that queries the nodes through a command/response protocol. All communication has to

go through this central bus controller, which queries its node in a cyclic pattern making all bus communication deterministic. Unplanned asynchronous communication is not supported [5, 6, 67].

On conventional copper cables the bus speed was limited to 1 Mbps. This has later been extended up to 20 Mbps on fiber optics in the latest Eurofighter Typhoon plane [5]. All configuration is preplanned in the bus controller at design time, therefore, no reconfiguration takes place in MIL-STD-1553. Besides a bus monitor that is passively listening to the bus, only a one bit parity check is added to the message that is used to determine its validity. If a node is found to be faulty, the bus controller will discontinue querying this node.

ARINC 629 Data Bus

ARINC 629 is built upon MIL-STD-1553B as a civil extension to the military bus. 629 does not require a central bus controller, which could be a single point of failure, but instead works in a decentralized way. A CSMA/CD style bus arbitration with dynamic time-slot allocation is used instead.

Specifically, all nodes sense the bus and wait for a defined synchronization gap (SG) after each transmission [5]. Afterwards, nodes wait a specific amount of time, called terminal gap (TG), depending on their priority till they start sending if the bus is free. High priority nodes have a short TG, lower priority nodes have a larger TG and will sense the bus as busy before sending and wait for the next synchronization gap. After each transmission cycle, nodes are prohibited to send for a backoff interval (TI). See Figure 3.4 for a graphical representation of this mechanism [68, 69].

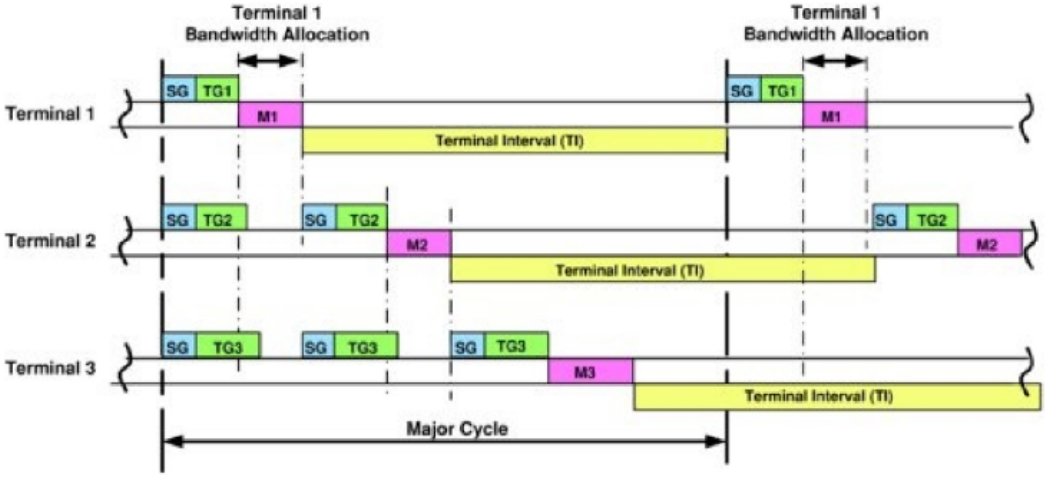


Figure 3.4: ARINC 629 message protocol (via [5])

ARINC 629 supports the redundant design of the network with quadruple and triple redundancy being supported allowing for fault masking. Each independent bus thereby offers a bandwidth of 1 Mbps. The system was not designed to be changed or reconfigured during operation and it is hard to dynamically do so due to its arbitration mechanism.

ARINC 429 Data Bus

ARINC 429 resembles one of the most used data buses in avionics. It is a simplistic bus that offers up to 100 kbps data rate, which can be achieved from a single source through a linear bus, with one directional communication only. For cases where bi-directional communication is needed, two buses are installed. The bus does not offer any kind of handshake or error recovery mechanism and is therefore not widely used anymore. For more information on this see [5].

3.2.2 Adopted Automotive & Industrial Networks

This sections presents real-time networks, which have not been used in aeronautical networks yet, however, showed promising features or use in other industries like automotive and could allow for future integration. One of the main points of this thesis is therefore the research of networks presented here and adding the missing parts that allows these networks to be considered for aircraft use.

3.2.2.1 IEC 62439-3 High-Availability Seamless Redundancy

High-Availability Seamless Redundancy (HSR) [70] as defined in IEC 62439-3 is a protocol for Ethernet networks. Its main goal is to provide seamless redundancy at an affordable cost, while being compatible with Ethernet. Packets coming from a node are duplicated before being sent out on two disjunctive paths across the network. This behavior ensures that even if one packet is lost, the second packet can still reach its destination. HSR uses the ring topology where packets are transmitted in opposing directions. No mechanism for traffic policing is specified in the current version leaving the network vulnerable to flooding.

Failure detection happens by checking Ethernet's CRC checksum, which checks for bit errors at every hop along the way. Faulty or lost packets will be transparently masked by the second arriving packet, making fault masking transparent for the end-device. Faults, however, will not be masked for any faults besides bit flips caught by the CRC.

HSR itself does not need any configuration, therefore no reconfiguration is supported either. Address look-up tables are filled dynamically, which makes them susceptible to address entry collisions.

Several works are available on traffic policing in HSR. Nsaif and Rhee [71] introduced a quick remove approach, that stops forwarding packets when the packet has been seen on the opposing link already. In the same paper, they described on how to use VLANs to minimize traffic on other network, e.g. coupled rings. They later extended their work with RURT [72] to minimize traffic in scenarios with coupled rings. While both works focus on minimizing traffic within the network, it does not offer any guarantees or traffic shaping. Lee et al. [73] proposed design methods for more reliable networks when using such coupled rings. Tien and Rhee [74] further proposed an extension to reduce redundant traffic.

3.2.2.2 IEEE AVB and TSN

First attempts of the IEEE Standards Association to give latency guarantees on Ethernet were done with IEEE Audio Video Bridging (AVB) aiming for a common standard allowing for real-time video transmission within a network. AVB consists of time synchronization services to allow frames from different cameras to be mixed, a rudimentary registration protocol for new video and audio streams as well as configurable shapers. While the shapers defined in the standard offer guarantees to individual streams, the mechanism behind it was later shown to be broken [75].

Besides the broken shaper mechanisms, AVB supported some amount of individual stream and upper latency bounds it was shown not to be sufficiently gradual for industrial use as it supported too few streams [76, 77]. Also analytical works compare Ethernet to IEEE AVB and how AVB can fit in industrial environments [76, 78, 79]. It was concluded that AVB does not offer enough advantages to replace Ethernet or a deterministic bus system and needs to be reevaluated with the AVB successor TSN.

Pushed by automotive for a common deterministic network, Time Sensitive Networking (TSN) was introduced by defining features in currently more than 30 separate standard documents [1]. Its main features include more robust time synchronization protocols, time-based scheduling, traffic policing, frame preemption, frame duplication for redundancy reasons and centralized and decentralized configuration protocols. An overview of TSN's features and capabilities is given in Chapter 4.3.

[1] http://www.ieee802.org/1/files/public/minutes/2017-03-closing-plenary-slides.pdf [Last accessed 2017/05]

3.2.2.3 OpenFlow

OpenFlow [80] is a communication protocol that controls a switch's forwarding behavior. It is running between a controller and multiple switches. Packets can be matched on different fields (i.e. destination MAC address) and then associated to an action or a set of actions. Actions include forwarding to ports, but also rewriting of certain parts of the packets like TTL or VLAN tags. In version 1.0 of the OpenFlow protocol the action set is modified by a single matching table entry and the action list associated to this match entry. In versions after 1.1.0 the action set is modified by instructions. Whenever a packet matches a table entry, one or several instructions are associated to this table entry. An instruction may then update, add or overwrite the action set for each packet.

Beginning with version 1.3.0 METER actions were introduced. A meter is for example a simple token bucket policer that can be instantiated and configured to a certain rate and burst. Whenever a flow exceeds the bucket's rate, the packet is dropped. If the packet complies with its traffic definition and the burst is not exceeded, the remaining actions in the action set will be executed. While the main application of meters is to rate limit packets sent to the controller it can also be reused to achieve quality of service behavior which is what is investigated in this section.

Sonkoly and Gulyás [81] proposed a framework to provide quality of service (QoS) in the Openflow enabled Ofelia testbed. The framework implements controllers that are aware of vendor-specific extensions of the switches to allow the use of different vendors.

Seddiki et al. [82] introduced FlowQoS with per flow bandwidth guarantees on a user's home router. It aims for an end-user style application where controllers dynamically install QoS rules on switches upon user requests for services like video or voice over IP with decisions and rules based on current bandwidth utilization. The solution is implemented by instantiating a two-switch virtual topology based on Open vSwitch and the use of different queues on the vSwitches. The switches are controlled by OpenFlow.

Egilmez et al. [83] proposed a new OpenFlow controller design supporting end-to-end quality of service. The controller continuously collects statistics from the switches to then dynamically place new requests on the correct path. As the controller only reacts on occurring congestion, such a monitor and react solution will not be able to handle hard guarantees.

He et al. [84] conducted a measurement study on OpenFlow switches, measuring the latencies between an incoming packet that is not matching any rule towards the controller and installation of rules. They concluded, that the time needed for installation of new

rules vastly depends on the implementation details of the specific switches and propose means on how to reduce the installation times.

Azodolmolky et al. [85] presented an analytical model based on network calculus to determine an upper latency and buffer size in a controller-controlled switch. The model assumes that packets have to go into the controller first and then be forwarded onto the switch. To prevent a single point of failure we assume to not have any controller interaction during operation, therefore our approach can be based on normal, OpenFlow unaware, network calculus.

Jacobs et al. [86] studied the use of Gigabit Ethernet switches in aircraft networks and concluded that the evaluated Gigabit Ethernet switches offer reliable low latency switching in over-provisioned or slightly under-provisioned scenarios.

Research regarding implementation of AFDX with COTS hardware has been performed in [87]. A commercial single-board computer was used running VxWorks with modified network drivers. The send and receive functions to access the driver were handled in a specific pseudo-partition of VxWorks, in order to allow communication from application partitions to the network. Only end-systems and no switches are considered.

3.2.3 Conclusion with Research Gap

The previous section showed the development of avionic networks from a distributed analogue network over distributed digital buses, federated digital system and finally to integrated modular systems with heterogeneous traffic requirements and reconfiguration demands.

For each of the networks, related work is presented with a technical summary in Table 3.2. The table categorizes the networks based on topology, supported kind of critical traffic, the configuration of it and its ability to withstand, isolate or mask faults. Furthermore, an assessment is given, whether the given network could be a candidate for a future avionic network.

ARINC 429 offers only low-speed unidirectional communication and therefore does not meet future requirements. Three other bus based systems were presented, ARINC 629, MIL-STD-1553 and TTP. Both ARINC 629 and TTP are used heavily in modern aircraft on the edge network connecting a large number of devices to the Ethernet backbone. While these buses will have their existence justified for a long time to come, they do not fulfill the bandwidth demands a future network should offer. MIL-STD-1553 is not common anymore due to its centralized communication master and its single point of failure.

All other networks presented use Ethernet as their physical layer technology. While TTEthernet aims to increase the bandwidth of TTP using Ethernet, it also increases configuration complexity of its TDMA like service as explained in Chapter 2.3.1.2 and has a large opposition in the aeronautic community. The current avionic solution is AFDX and offers determinism and redundancy features desired, however, it is not a widely used standard outside aeronautics and is not configurable during runtime nor supporting any kind of mixed traffic.

Promising candidates for a future avionic network are HSR, AVB/TSN and OpenFlow, that are used in either automotive, provider backbones or research. While AVB has some desired features, it only supports up to 13 streams in its highest service class [88] and has shown to overshoot its promised latency claims in certain worst case cases [75]. HSR has promising redundancy features and an ease of configuration, however, lacks deterministic features and support for mixed criticality. TSN is still in the standardization phase and not finished at the time of this writing. While it might offer all features needed, it will be too complex and too feature rich to get through certification, therefore, a smaller avionic feature profile is proposed in this thesis as well as a preliminary evaluation of its use in aircraft. Similarly, OpenFlow offers advanced features in out of band reconfiguration, however, lacks deterministic features.

In the next chapter of this thesis, we focus on the three mentioned candidates and how to improve critical points to make them ideal candidates for future avionic networks.

			Support													Assessment		
			Topology				Traffic			Configuration			Fault					
Real-Time Network	Speed	Related Work	Shared Bus	Star Network	Ring Network	Mesh Network	Time-Triggered	Rate-Constrained	Best Effort	In-Band Configuration	Online Reconfiguration	Vendor Independent	Fault Detection	Fault Isolation	Fault Masking	Mass Industry Usage	Aircraft Certifiable	Suitability for FAN[2]
AFDX *(ARINC 664)*	100 Mbps	[40, 53–60]	n	y	n	y	n	y	n	n	n	n	y	y	y	o	+	o
TTP *(SAE AS6003)*	5/25 Mbps	[6, 61–64]	y	y	n	n	y	n	n	y	n	n	y	y	n	o	+	-
TT Ethernet *(SAE AS6802)*	agnostic[3]	[51, 65, 89]	n	y	n[6]	n	y	y	y	n	n	n	y	y	y	-	o	o
MIL STD 1553B	20 Mbps	[5, 6, 67]	y	n	n	n	y	n	n	n	n	n	y	n	n	o	+	-
ARINC 629	2 Mbps	[5, 68, 69]	y	n	n	n	n	y	y	n	n	n	y	y	y	-	+	-
ARINC 429	100 Kbps	[5, 6]	y	n	n	n	n	n	n	n	n	n	n	n	n	o	+	-
HSR *(IEC 62439-3)*	agnostic[3]	[70–74]	n	n	y	y	n	n	y	y	n	n	y	n	y	o	-	o
AVB *(IEEE 802.1)*	agnostic[3]	[78, 79, 88]	n	y	y	y	n	y[4]	y	n	y	y	y	y	n	o	o	-
TSN[5] *(IEEE 802.1/3)*	agnostic[3]	[90]	n	y	y	y	y	y	y	n[6]	y	y	y	y	y	+	o	o
OpenFlow	agnostic[3]	[80–87]	n	y	y	y	n	n[6]	y	n[6]	y	y	y	n	n	o	o	o

Table 3.2: Overview of related work; with supported(y), unsupported(n); positive assessment (+), average(o) and low(-)

[2]Future Avionic Network
[3]Uses Ethernet as Physical Layer
[4]Only small number of streams
[5] Features expected once standardized
[6] Not per Standard, but can be implemented

3.3 Self-Configuration

This section gives an overview of related work in the area of self-configuration mechanisms for real-time networks. In the last years multiple projects with the aim of real-time self-configuration have been proposed. Their main distinction points are the underlying networks and their support for fail-over as well as support for multicast traffic. We give an overview of our findings with current research gaps in Table 3.3.

3.3.1 Management Data & Configuration Channels

Certain data models are used within each device to store information about the device itself, its configuration state, operational state and statistics. Further each device's data model can include information about its neighbors or connected devices and attributes of those. Such data models are usually accessible via the network to allow a coordinating instance to detect the network state.

Management Information Base (MIB) [91] describes modules and structures of managed objects and defines unique identifiers. Such identifiers can then be used to query the data via the Simple Network Management Protocol (SNMP). A new and easier readable data model is described with the Yet Another Next Generation (YANG) language [92]. This language is currently used to describe most IEEE and IETF protocols to ease configuration across devices and can be queried/written by the Netconf protocol.

OpenFlow (see Chapter 3.2.2.3) is a standardized protocol to control switch forwarding planes. While it was not designed for administrative actions like querying link-status, it can still be used to control the policing behavior as well as reading out packet statistics. One or several controllers update and receive management data from switches and react to it.

Wisniewski et al. [93] introduces a concept for seamless reconfiguration of Profinet Time Triggered networks. By using Profinet's synchronous and asynchronous phases, a channel not disturbing the operational traffic is created. This channel is then used to exchange configuration information. The work does not specify how this is done or which protocol is envisioned. A fail-over strategy is not considered.

3.3.2 Topology Discovery

Link Layer Discovery (LLDP) [94] is a data layer protocol to discover neighbors. End-devices and switches broadcast their current configuration including an identifier and a

management address. In an extension called LLDP-MED [95] additional parameters for end-devices were specified like power needs. LLDP messages are not forwarded between switches and their information is stored in the local MIB. To build up a topology view all managed objects in the network have to be queried.

Based on LLDP, this concept has been extended to the OpenFlow Discovery Protocol (OFDP) with its optimized version OFDPv2 [96]. A controller floods LLDP messages across the network and receives them at the next OpenFlow enabled switch that generated a packet-in event to then build up a topology overview.

Intermediate System to Intermediate System (IS-IS) [97] is a link-state routing protocol that intends to build a network-wide shared database in each node. Nodes are flooding link state information across the network to aggregate the flooded information. Dijkstra's algorithm is then used to compute the best path through the network.

Ochoa-Aday et al. [98] describe a new way of doing topology discovery in software-defined networks. Their approach Software-Defined Topology Discovery Protocol (SDTDP) is based on agents, that are locally installed in every switch. The agents allow for decentralized discovery of topology by comparing delays between neighbors. While this approach works for the failure-free case, during faults it will not give reliable results. Further, it was not evaluated how to account for changes in delays due to load and queue delays.

3.3.3 Active Topology Control

The *Spanning Tree Protocol* (STP) is used to find a loop-free tree over a meshed layer 2 network. Later it has been enhanced to support multiple trees (MSTP) starting in each sink. In STP, links are disabled to prevent network loops.

IS-IS Path Control for Reservation (PCR) is currently in discussion in the IEEE TSN working group 802.1Qca, to make up for short-comings of Shortest Path Bridging (SPB). It aims for support of redundant paths within a layer 2 network. This standard is expected to be completed by end of 2017.

Egilmez et al. [83] propose with OpenQoS a new OpenFlow controller supporting soft end-to-end quality of service. The controller continuously collects statistics from the switches to then dynamically place new requests on underutilized paths. The controller reacts to the occurrence of congestion and does not apply meters.

Sharma et al. [99] present a fail-over approach for OpenFlow networks. The central controller reacts to link change events sent by the switches and calculates new available

paths. Restoration takes place in the millisecond range. However, merely reacting to failures still means packet loss, which is not wanted for real-time networks. In [100] the author extended the work to support in-band configuration that removes the need for an out-of-band network to connect OpenFlow switches to the controller.

3.3.4 Stream Reservation

The *Stream Reservation Protocol* (SRP) [101] consists of a layer 2 registration and reservation protocol. The registration protocol is initiated by listeners. The reservation protocol is triggered by registration and deregistration events. Currently standardization work is going on to add support for more streams, have configurable stream reservation classes and a deterministic stream reservation convergence. This work is done within 802.1Qcc as part of the TSN working group and expected to be finalized around 2018. Path finding or handling is not included in SRP.

Kammerer and Obermaisser [102] present a mechanism to dynamically configure a CAN network. Their CAN router sits in the middle of a CAN star network and accepts configuration requests from multiple CAN sources and guarantees a safe forwarding through the router. The implementation can stop messages that are outside their traffic specifications, however, does not offer restoration due to the star topology.

Farzaneh and Knoll [103] present an ontology based plug and play mechanisms for IEEE TSN's Time Aware Shaper (TAS). Each end-device sends a device profile to a device discovery service, which then registers the device with an application management service that configures the network. It is not described how the service configures the TAS or if the end-devices have to be time synchronized. A simple star architecture is used, therefore, no reconfiguration or fault recovery is considered.

3.3.5 Self-Configuration

A classic configuration protocol is the *Dynamic Host Configuration Protocol* (DHCP) [104]. DHCP dynamically distributes network configuration parameters on a layer 2 network. A central server keeps the state on which devices were configured. This idea was later extended to a decentralized approach [105, 106] without a central server, where nodes sense the network for collisions before taking ownership of an IP address.

Dürkop et al. [107] propose an approach for auto-configuration of real-time Ethernet systems. While they present a case study for Profinet IO, such a concept could also be applied to other real-time Ethernet systems. However, the approach lacks support for

multicast traffic as well as reaction to failures during operation. Later they extended their work in [108] by introducing an ad-hoc channel and a middleware.

Reinhart et al. [109] present a framework and general concept for a *Plug & Produce* network. A central configuration manager module queries devices for XML configuration files to compile a working view of the services in the network. The manager then disables the basic network communication and switches to an operational mode. Reconfiguration during runtime or reaction to failures is not possible.

Imtiaz et al. [110] introduce a means to auto-configure MAC addresses in line or tree topologies which can then be used for further bootstrapping. The approach does not consider online reconfiguration and is static in the way, that it is only executed once at bootup.

3.3.6 Conclusion with Research Gap

The previous section gave an overview of the related work in the area of self-configuration mechanisms. The works were divided into the categories management protocols, topology discovery and control as well as stream reservation and other self-configuration mechanisms. A full overview and categorical comparison including research gaps is shown in Table 3.3.

Main distinction points between the works is the support of reconfiguration at run-time or only at boot-time. Reconfiguration is important to allow reaction to failures.

Reinhart et al. [109] present solutions that allow devices to be discovered during bootstrap in the industrial networks Powerlink and Profinet. However, due to the nature of the used real-time networks a reconfiguration is not feasible with their concepts.

Farzaneh and Knoll [103] propose solutions that can offer hard real-time guarantees but only work on star topologies and do not support COTS devices.

Egilmez [83] and Sharma [99] present solutions based on OpenFlow, that due to the underlying network do support COTS devices. Egilmez also offers soft quality of service guarantees by rerouting traffic depending on the network load.

Concluding from the related work, it can be seen that no solution offers all features that are needed, namely a plug and play mechanism supporting reconfiguration that is based on a real-time network which supports COTS devices as well as purpose-built devices.

		Configuration Process					Operational Traffic			
Related Work	Technology	In-Band Management	Online Service Discovery	Reconfigurable at run-time	Isolation from Data Traffic	COTS Support	QoS Guarantees	QoS Guarantees for COTS	COTS Support	Suitability for FAN[7]
ARINC 664 (Cha. 3.2.1.1)	AFDX	n	n	n	n/a	n	y	n	y	o
OpenFlow (Cha. 3.2.2.3)	OpenFlow	n	n	y	n	n	n	n	y	o
Reinhart et al. [109]	Powerlink	y	y	n	n/a	n	n	n	n	o
Dürkop et al. [107][108]	Profinet	n [8]	y	n	n	n	h[10]	n	n	+
Farzaneh and Knoll [103]	IEEE TSN[9]	y	y	n	n	n	h[10]	n	n	o
Kammerer and Obermaisser [102]	CAN	n/a	y	y	n	n	h[10]	n	y	-
Wisniewski et al. [93]	Profinet	y	n/a	y	y	n	h[10]	n	n	o
Egilmez et al. [83]	OpenFlow	n	n	y	n	y	s[11]	s[11]	y	o
Sharma et al. [100]	OpenFlow	y	n	y	n	y	n	n	y	+

Table 3.3: Overview of related self-configuration mechanisms;
with supported(y), unsupported(n); positive assessment (+), average(o) and low(-)

[7] Future Avionic Network
[8] In loop-free Networks
[9] Time Aware Shaper (TAS) only
[10] Hard Quality of Service (QoS) guarantee
[11] Soft Quality of Service (QoS) guarantee

Chapter 4

Future Avionic Networks

There is a growing trend for real-time Ethernet in nearly all industrial branches. Each industry has its own Ethernet-based protocols, as shown in Chapter 3.2, where none of the networks offers a solution to all problems. Such networks are used for critical systems that need to work even in case of failures and offer a maximum of safety, reliability and security. While safety translates to having no critical failure in the system, critical systems also need to have a guarantee on a network level, that packets are delivered in all cases. Therefore, the network must ensure that packets are delivered before a deadline. When speaking about deadline guarantee for packets, often the term deterministic is used. For a network architecture to be called deterministic, it must fulfill the following points: (a) formal verification of maximum end-to-end latencies and (b) mechanisms in the network to guarantee that ill-behaved end-systems will not interfere with well-behaved end-systems.

In this chapter we present our work that aims to close research gaps found in Chapter 3, a combination of determinism, reliability and plug and play support in networks . The fundamental design space that was identified in Chapter 3 spans in between synchronized vs. non synchronized mechanisms in networks, optimization of topology and the use of COTS switches. We propose here three solutions in Sections 4.1, 4.2 and 4.3 that are the most promising instances of this solution space while taking into account aerospace specific requirements analyzed in Chapter 3.1. The works have been presented at various scientific conferences, as indicated at the beginning of each sub-chapter.

Structure of this chapter

In Section 4.1 we start this chapter with a deterministic high availability ring network. Section 4.2 applies a different concept to enable deterministic OpenFlow enabled switches. The next Section 4.3 looks at an upcoming IEEE deterministic network, presents a simulation framework for the evaluation and presents an aerospace-industrial profile of the standard which specifies the specific use of the multitude of sub-standards. Finally we compare results and give an evaluation of the presented solutions in Section 4.4.

4.1 Priority-Based Forwarding in Switches with a Global Timebase

Note This section is part of a publication presented at 8th International Workshop on Communication Technologies for Vehicles (Nets4Cars/Nets4Trains/Nets4Aircraft) in 2015 [111].

In this subchapter we present an extension to High Availability Seamless Redundancy (HSR) [70] (see Chapter 3.2.2.1). While HSR offers support for plug and play devices and is therefore easily configurable, it lacks any kind of real-time features. We thus present here an extension to HSR, that can guarantee packet arrival within a certain time while still offering best-effort traffic in order to support mixed criticality traffic. Furthermore, a mathematical model is developed, that describes the upper latency limit of the proposed mechanism. We compare the proposed extension of HSR to the original version, using a network simulation framework to show the advantages and weaknesses.

4.1.1 Definitions

Delay Variability

In HSR, packets are always sent twice, therefore for determining a maximum latency both paths have to be considered. If always this maximum case is considered, no difference will be seen in case of a failure compared to the fault-free scenario. The worst case will thus be created, when a direct neighbor is addressed: a short path with a hop count of 0 and a long path across the whole ring. The time between the arrival of both packets is called *delay variability* V. In order to minimize the maximum latency, the delay variability time has to be minimized. This can be done by relaxing (deprioritizing) the short path while at the same time prioritizing the longer path. An illustration of this can be seen in Figure 4.1. While the short path latency allows for several strategies, a minimization of the longer path is achieved by giving priority to packets that are already on the ring. We, therefore, opt for the ring-first scheduling strategy as it is easy to implement in hardware and keeps buffering of packets on a ring to the minimum.

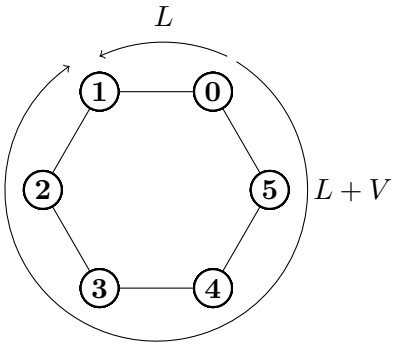


Figure 4.1: Delay variability on a ring network

Time-Aware Queue

To police incoming traffic and keep latency to a minimum, we propose the use of *Time-Aware Queues* (TAQ). A TAQ is a normal FIFO queue that is enabled by a time-based schedule. Its schedule works on a global time-base and allows the dequeuing of packets from its queue only when the schedule allows for it. We introduce one or more TAQs connected along with several normal queues to a *strict priority scheduler* (SPQ). The SPQ gives the highest priority to the TAQs; the second highest priority to the incoming ring traffic and the lowest priority to local best-effort traffic. As the TAQs will never send at the same time, their priority can be considered to be the same. An illustration of this can be seen in Figure 4.2. Because nodes only need to know their local schedule and treat other ring traffic in normal queues, this mechanism is enabling a decentralized TDMA while the ring traffic is still going first.

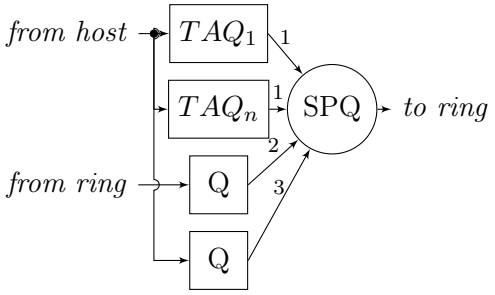


Figure 4.2: Time-aware queues (TAQ) in each node port

4.1.2 Mathematical Model

In this section, we develop a mathematical model to describe upper limits of traffic latencies for the time aware queues defined in Section 4.1.1. It is based on the synchronization of queues which, however, in case of faults can be lost. While TSN [1] and TTEthernet

[65] try to prevent such a failure mode by using redundant clock masters, the HSR standard does not cover such mechanisms and is as such vulnerable to synchronization loss. Therefore, the model aims to give an upper bound in a synchronized condition as well as under a clock failure.

We introduce the following naming conventions, following the definitions in RFC 1242 [112]. An exemplary path in Figure 4.1 from node 0 to node 1 can take two paths. The first path with the lesser amounts of hops is called the *short path* and has in this example a hop count of 0. The longer path is called *redundant path* and has a hop-count of 4 in the example. We assume a static processing delay in each hop of $T_{switching}$.

4.1.2.1 One TAQ per Node

In static offline configured networks, often called engineered networks, network designers typically specify a refresh time that describes how often an information needs to be updated. We call this time T_{cycle} and it is assumed to be given by the scenario. Also, the designers specify how much data D needs to be transferred. Therefore, the slot per data-flow is given by $T_{slot} = \frac{D}{BW}$.

Each time a packet is transmitted, the physical line might already be in use causing a delay of $T_{cross} = \frac{D_{Cross}}{BW}$ called *head-of-line blocking*, when a packet of lower priority has to finish transmission first. Because we want to allow mixed criticality and thus best-effort traffic, we have to assume D_{Cross} to be the maximum allowed Ethernet packet size and therefore have a static T_{Cross} depending only on the used transmission speed BW.

While the TAQ is sending according to its schedule, the local application in the node is not aware of the time synchronization. Therefore, the host sends packets at arbitrary points in time to the network card where they will be transmitted when the schedule permits. This maximum local delay is, therefore, expressed by T^N_{local} for a node N with T^N_{Cycle} representing the individual cycle length of that node:

$$T^N_{\text{local}} = T_{switching} + T_{Cross} + T^N_{Cycle} \tag{4.1}$$

This worst-case is created, when the packets arrive at the network card right after the slot has ended and have to wait a whole cycle before the slot arrives again thus incurring a delay of $T_{cycle} - T_{slot}$. The transmission of the packet itself takes another T_{slot} so that it can be removed from the equation.

Time spent at each additional host resembles the interfering slots of that host and its head-of-line blocking and is expressed with T^N_{perhop} for a node N:

$$T^N_{\text{perhop}} = T_{switching} + T_{Cross} + T^N_{Slot} \tag{4.2}$$

The maximum end-to-end delay E from a node N across the ring is going through a multitude of hops on its path. All hops on the path are added to the set PATH. The end-to-end delay E from a node N via all nodes in the set PATH is then given by Equation (4.3). T^N_{Slot} is added to follow the RFC 1242 definition of latency in an empty channel with T^A_{perhop} representing interference.

$$E^{N\rightarrow\text{PATH}} = T^N_{\text{local}} + \sum_{A\in\text{PATH}} T^A_{\text{perhop}} + T^N_{Slot} \tag{4.3}$$

While Equation (4.3) describes the case that happens under untimely scheduled cross-traffic, T^A_{Slot} will not interfere at each hop if synchronization is working. Therefore, E can be simplified to the following form.

$$E^{N\rightarrow\text{PATH}}_{\text{synchronized}} = \underbrace{T_{switching} + T_{Cross}}_{T^N_{\text{local}}} + \text{size(PATH)} \cdot (\underbrace{T_{switching} + T_{Cross}}_{T^A_{\text{perhop}}} + T^N_{Slot}) \tag{4.4}$$

4.1.2.2 Multiple TAQs per Node

To allow multiple applications per node, it is likely that different schedules are needed as each application might have its own transmission cycles. Therefore, we introduce support for multiple TAQs per node each with its own schedule. The schedules have to be globally synchronized so that no collisions occur.

We extend the previous formulas with an additional argument i indicating the TAQ on the node, denoted by $T^{N,i}_{Cycle}$ and $T^{N,i}_{Slot}$. Equation (4.1) is extended as follows for a given TAQ i on a node N:

$$T^{N,i}_{\text{local}} = T_{switching} + T_{Cross} + T^{N,i}_{Cycle} \tag{4.5}$$

Again $T^{N,i}_{\text{local}}$ denotes the maximum delay that can occur on the local node. As the schedules for parallel queues have to be configured collision-free, other queues do not influence this maximum value.

$$T^N_{\text{perhop}} = T_{switching} + T_{Cross} + \sum_{i=1}^{\#\text{TAQs of N}} T^{N,i}_{Slot} \tag{4.6}$$

This bound could be lowered if the unused and available time between slots is taken into consideration. Following Equation (4.5) and (4.6), the maximum end-to-end delay E across a set PATH containing all nodes on the path is as follows:

$$E^{N,i\rightarrow\mathrm{PATH}} = T^{N,i}_{\mathrm{local}} + \sum_{A\in\mathrm{PATH}} (T^{A}_{\mathrm{perhop}} + T^{N,i}_{Slot}) \tag{4.7}$$

For $E^{N,i\rightarrow\mathrm{PATH}}_{\mathrm{synchronized}}$ the same simplifications as before can be made leaving only a dependence on $T_{switching}$, T_{Cross} and $T^{N,i}_{Slot}$ as shown below.

$$\begin{aligned} E^{N,i\rightarrow\mathrm{PATH}} = & \underbrace{T_{switching} + T_{Cross} + T^{N,i}_{Cycle}}_{T^{N,i}_{\mathrm{local}}} \\ & + \sum_{A\in\mathrm{PATH}} \underbrace{T_{switching} + T_{Cross} + \sum_{i=1}^{\#\text{TAQs of A}} T^{A,i}_{Slot}}_{T^{A}_{\mathrm{perhop}}} \\ & + \sum_{A\in\mathrm{PATH}} T^{N,i}_{Slot} \end{aligned} \tag{4.8}$$

4.1.2.3 Specifics to High Availability Seamless Redundancy (HSR)

In a HSR ring with nodes n ∈ NODES, the packets always take two paths. Therefore we can specify a HSR end-to-end latency H for the short path, which is usually taken, and the redundant path, which has to be considered in case of a link failure.

$$H^{N\rightarrow\mathrm{PATH}} = \begin{cases} E^{N\rightarrow\mathrm{PATH}} & \text{Short Path} \\ E^{N\rightarrow NODES\setminus\{\mathrm{PATH}\cup N\}} & \text{Redundant Path} \end{cases} \tag{4.9}$$

Following the assumption of being able to handle one failure per containment region, the maximum latency is determined by the link down case. As the packets in HSR take two different path, the two HSR copies take different times to arrive at the destination. The maximum packet end-to-end delay variability V of those two copies is described in the Equation (4.10), where the longer path is assumed to miss its slots due to synchronization errors while the shorter path has a working synchronization. PATH represents here the short path's hops.

$$V^{N\rightarrow\mathrm{PATH}} = E^{N\rightarrow NODES\setminus\{\mathrm{PATH}\cup N\}} - E^{N\rightarrow\mathrm{PATH}}_{\mathrm{synchronized}} \tag{4.10}$$

4.1.2.4 Gain of Time-Aware Queues

To make up for the complexity of time-awareness a major advantage should be enabled. As a reference for comparison we use a non-synchronized network like HSR in combination with a simple traffic shaper. The resulting latency is highly dependent on cross-traffic. Best-effort cross-traffic will have the same effect on both mechanisms and is as such not neglected in the following.

HSR Model

We use Equation (4.5) to compare it to the non-synchronized mechanism assuming the worst-case of just having missed the slot. While the equation gives a dependence on the cycle, it also limits the amount of slots that can be inserted.

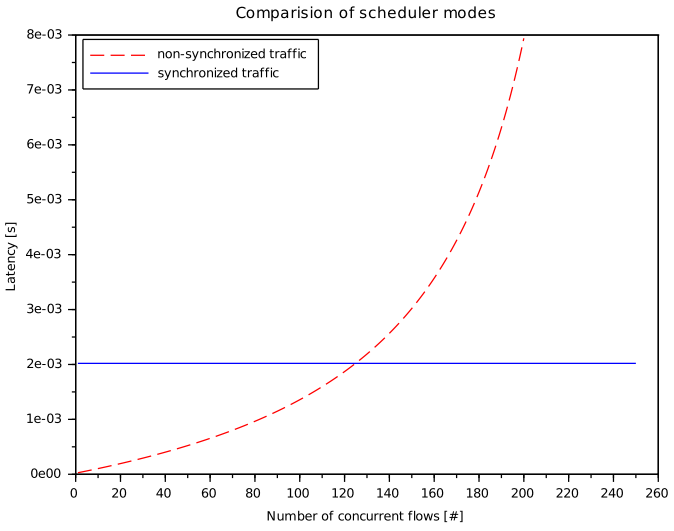


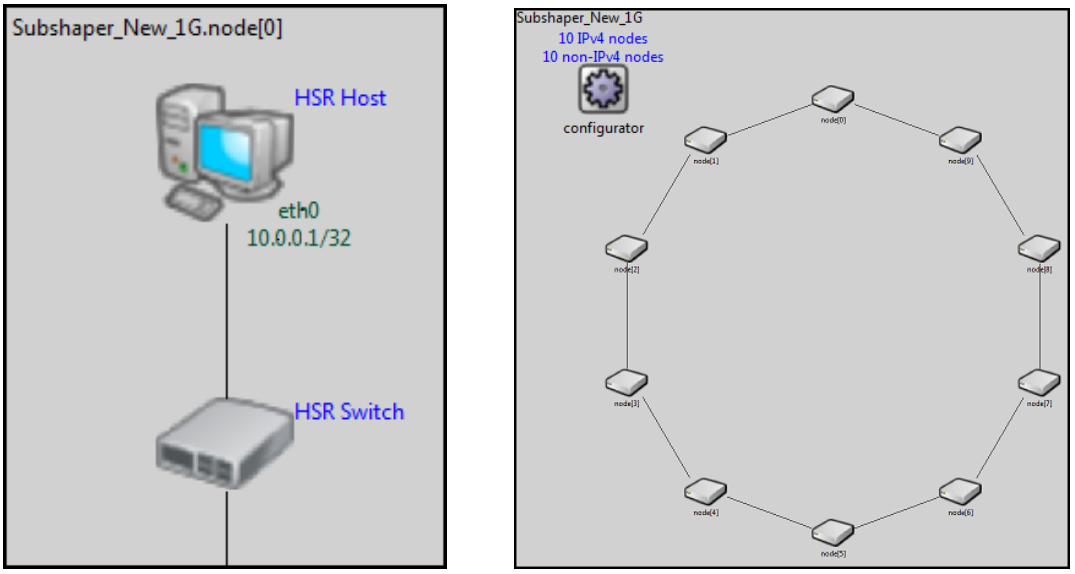
Figure 4.3: Comparison of non-synchronized and synchronized traffic
packet size = 100byte, transmission speed = 100Mbps, switching time = 20μs, cycle time = 2ms

In Figure 4.3 it can be seen, that non-synchronized flows can send right away but have an increasing latency with each additional flow being added. This gives an advantage to the time-aware architecture, where the network card needs to wait for the cycle. For the scheduled traffic, however, the resulting latency is then not dependent on other flows. Further the synchronization gives a better performance above a certain threshold. However, only a certain number of slots can be assigned to a cycle. If more flows are to be added, larger cycle times need to be selected giving a higher latency.

4.1.3 Simulation of Extended HSR

To simulate the proposed features we implemented a network simulation framework to then compare the worst-case bounds of the analytical model with actual delays in example scenarios in the simulation. We implemented HSR according to IEC 62439-3:2012 [40] in a simulation environment including its main features of packet duplication and deduplication. The simulator used is OMNeT++ [113] with the INET extension [114].

The HSR standard specified a profile for the *Precision Time Protocol* (PTP). PTP is a time synchronization protocol, that achieves sub-microsecond accuracy between clocks. For PTP no communication model has been implemented, however, a clock model is introduced following the accuracy descriptions demanded in the standard with an accuracy of 50ppm and a maximum drift of 1ppm/s. An example node and network implementation can be seen in Figure 4.4.



(a) HSR Node in OMNeT++ (b) HSR Ring Network in OMNeT++

Figure 4.4: Examples of our OMNeT++ network simulation

4.1.4 Comparison of Mathematical Model and Simulation Results

In this section we show some of the simulation results that have been generated with the simulation environment from Section 4.1.3 and compare it to the analytical model. In the following all nodes are configured the same way. Each node has one time-triggered stream and a variable amount of best-effort traffic to random destinations across the ring. The topology used is a simple single ring with 10 nodes as shown in Figure 4.4b. Further the parameters from Table 4.1 were used in the simulation.

Abbr.	Parameter Description	Value
N	Number of nodes in the network	10
$T_{switching}$	Static switching delay	$0\mu s$
T_{cross}	Best-effort cross traffic	$12\mu s$
T_{slot}	Slot length for each node	$5\mu s$
T_{cycle}	Cycle length for each node	$1000\mu s$
BW	Transmission speed	$1Gbps$
$Load$	Node's use of bandwidth for BE (1 equal to: $1 \cdot \frac{BW}{N}$)	1

Table 4.1: Simulation parameters for the OMNeT++ network simulation (Note: $T_{switching}$ and BW have been chosen as seen above to simulate the worst-case with max load on the network; the simulation itself supports arbitrary settings here.)

4.1.4.1 Normal Working Mode without Failure

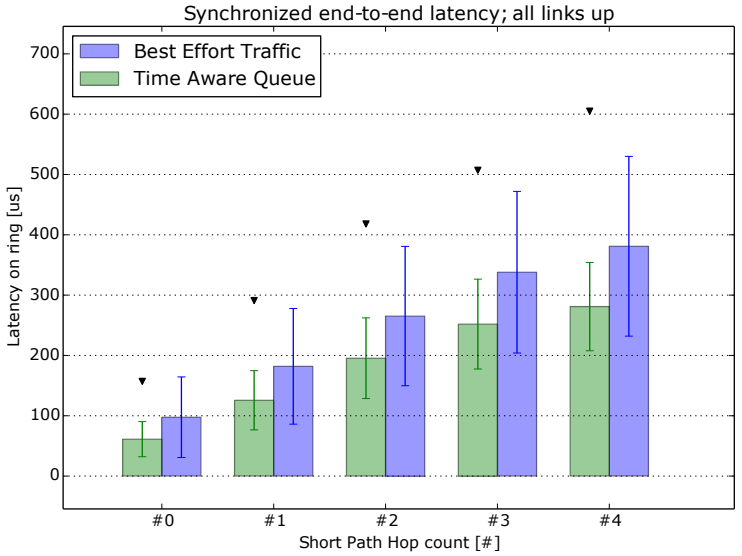
In a first step we simulated a HSR ring without any failures. It can be seen in Figure 4.5a, that in normal operating mode the network behaves as expected. Messages that have to cross more hops have a higher latency. Further, it can be seen that synchronized traffic experiences a lower latency than best-effort traffic. Best-effort traffic achieved on average also a fair latency, mainly dependent on the strategy for forwarding packets and the utilization of the ring.

4.1.4.2 Single Link Failure Case

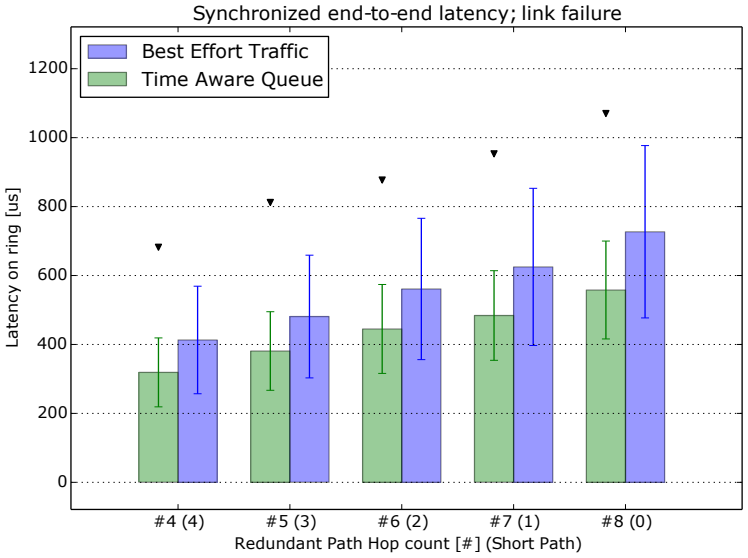
In a next step we looked at HSR's main feature and the resilience against link failures. Therefore, the shortest path was broken so that the slower redundant message needs to be waited for. In Figure 4.5b it can be seen, that messages on the previous shortest path to a direct neighbor now take about 8 times longer to arrive than before. Messages that have to cross 4 hops like before experience around the same latency. The load factor on the ring itself does not increase as the same messages are transmitted in case of normal and failure operation.

4.1.4.3 Loss of Time Synchronization

One failure mode that can appear is the loss of time synchronization. Therefore, we manually construct a worst case with a maximum of cross-traffic and compare the observed results with the latency computed with the analytical model. In Figure 4.6 it can be seen, that the cycle-miss time is higher than even a link failure on the same path.



(a) All links running end-to-end latency



(b) Link broken end-to-end latency

Figure 4.5: Simulated vs analytic latency for running and failure case
Experiment run 50 times; bars indicate standard deviation;
black arrow indicating maximum value predicted by model

Exemplary we show here the calculation for the time-sync loss case. The model predicts:

$$\begin{aligned} E^{8\to\{7,6,5,4,3,2,1\}} &= T^{8}_{\text{local}} + \sum_{A\in\{7,6,5,4,3,2,1\}} (T^{A}_{\text{perhop}} + T^{8}_{Slot}) \\ &= T^{8}_{Cycle} + T_{Cross} + 7\cdot(T_{Cross} + T^{8}_{Slot}) \\ &= 1000\mu s + 12\mu s + 7\cdot(12\mu s + 5\mu s) = 1131\mu s \end{aligned}$$

The maximum value in the simulation is given by $1048\mu s$ and is thus bound by the model.

The difference can be explained by not perfectly aligned traffic that would require also the 12μs. The model also holds if packets on the short path are lost due to, for example, a link failure.

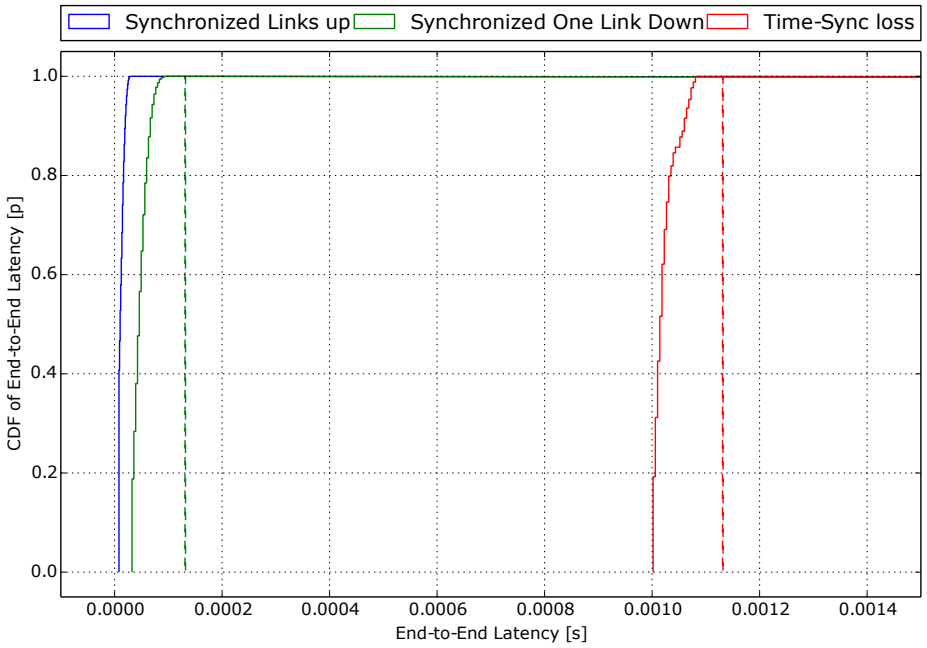


Figure 4.6: Cumulative distribution function (CDF) of latency of in normal, link fail and time-sync loss case for simulation and analytical model (dashed line)

4.1.5 Conclusion

In this subchapter an OMNeT++ network simulation of HSR has been presented. A mechanism to guarantee packet arrival was developed and implemented in the network simulation. Further, a mathematical model for the mechanism has been developed and compared with the simulation results. It was shown, that time-awareness offers better latencies in higher load scenarios. However, it was also shown that the loss of time-awareness increases the latency significantly and it has to be carefully evaluated, if a time based system were to be introduced in critical avionic networks.

4.1.5.1 Open Problems and Future Work

One failure mode that has to be considered during certification is the babbling idiot case of a switch. A crashed switch would be stopped by HSR's deduplication mechanism: ring nodes shall dispose duplicates of packets that arrived with the same sequence-number and source MAC. This, however, is only true if the sequence numbers within such frames do not increase or appear randomly which cannot be guaranteed in a failure case. Also, as

the address space for MAC-addresses is quite large, hardware implementations usually do some kind of hash-table look-up which is not certifiable due to potential collision problems. Another problem exists in avionics, where one design guideline is to always have a second line of defense, meaning a second mechanism independent of the first to stop such behavior.

4.2 Real-Time OpenFlow Network with Non-Synchronized Switches

Note This section is part of a publication presented at 2nd International Workshop on Management of SDN and NFV Systems which was hosted at 11th International Conference on Network and Service Management (CNSM) in 2015 [115].

In this subchapter we present a concept on how to achieve real-time behavior using standard commercially available Software Defined Networking switches supporting OpenFlow [80] (see Chapter 3.2.2.3). The contributions are twofold: we show how to achieve bounded latencies with OpenFlow by applying metering on a rule base for each packet and second we implement and evaluate the concept on an *HP E3800* COTS switch. We then introduce a mathematical model and present synthetic measurements with the COTS switch and finally compare the behavior to a state of the art AFDX switch currently in use in aircraft.

4.2.1 Definition & Concept

The way bounded latencies are achieved in AFDX is based on two steps: (1) Mapping of each incoming packet to a specific flow based on its MAC destination address and (2) assuring conformance to a predefined traffic behavior for this flow. If a packet cannot be mapped to a flow in (1) or exceeds the rate allowed in (2), the packet will be dropped before entering the switch's backplane. This way only conformant packets are allowed in and with knowledge about the switch's line speed the maximum queuing time and buffer size can be calculated by using network calculus.

4.2.1.1 Token Bucket in AFDX and OpenFlow

The OpenFlow 1.3 [80] standard allows to instantiate meters with different types. One of the types in the standard is a simple rate limiter called *OFPMBT_DROP*. The DROP band models a token bucket behavior which allows to drop any packet that exceeds a certain flow's rate. The band can calculate this rate either in kilobits/sec or in packets/sec.

The DROP band is based on the token bucket in Figure 4.7 which works as follows. Tokens get added at a specific token rate r into a bucket with the size b. Whenever a packet arrives at the token bucket, it is allowed to consume all the tokens in the bucket.

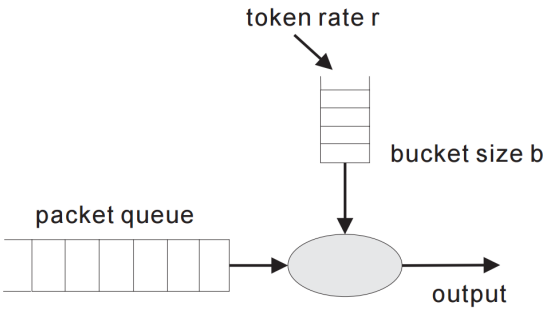


Figure 4.7: Token Bucket Mechanism

If not enough tokens are available, the packet can either be delayed or dropped. For a bounded latency packets will be dropped, as packets not following their traffic contract will likely not do so in the future. The advantage of token buckets against other schedulers is its independence of a global time base as well as allowing for burstiness in the traffic behavior.

Mathematically the traffic constrained by a token bucket can be modeled using network calculus [40] (NC). In NC traffic flows are modeled as $\alpha_{r,b}(t) = (rt + b)$ for $t > 0$ with $r = \frac{s_{max}}{BAG}$ and $b = s_{max}$ and s_{max} being the maximum packet size. A typical service curve β for a switch or router will usually look like $\beta_{R,T}(t) = R\lceil t - T \rceil^+$ with R being the output port bandwidth and T the static processing delay.

By applying NC the traffic characteristic of the output flow α^* through a switch offering a service curve β can then be calculated according to

$$\alpha^* = \alpha \phi \beta(t) = \sup_{u \geq 0}\{\alpha(t+u) - \beta(u)\} \tag{4.11}$$

4.2.1.2 Deterministic Behavior and End-to-End Latencies

The above mechanism explains how to describe a single system. The service curve experienced through a series of concatenated systems can be described as $\beta_{R1,T1} \otimes \beta_{R2,T2} = \beta_{min(R1,R2),T1+T2}$. With that an end-to-end delay bound D_0 can be calculated as

$$D_0 = \frac{b}{R} + T \tag{4.12}$$

with $R = \min_i(R_i)$, $T = \sum_i T_i$ and b the burst of the arrival curve and i being the index over the concateneated systems. This formula is known as the *Pay Bursts Only Once* theorem [40].

Besides the calculated traffic no other traffic is allowed on the network and all devices have to follow their traffic contract. Devices sending more packets than allowed will experience packet drops at the next policed ingress port. Such behavior is acceptable, as such devices can be considered broken and are not likely to be fixed by any means of the network itself.

4.2.1.3 Matching of Virtual Links

The matching of virtual links is based on the packet's destination MAC address. OpenFlow allows for exact and wildcard matching of destination MAC addresses. To separate the flows from each other an exact matching entry for each MAC address is needed, each mapped to an individual meter. To allow for AFDX's multicast behavior, multiple actions can be associated to each entry, e.g. by having multiple output actions.

4.2.1.4 Role of the OpenFlow Controller

OpenFlow allows for the installation of rules during the run-time of a switch. Such a behavior is not desired in avionics, especially not during flight. While a single point of failure could be prevented by using multiple controllers, we assume the controllers only to install rules during a maintenance phase on ground and then leave all configuration information as-is until the next maintenance. The controller will therefore be turned off during flight. The OpenFlow standard already supports this option by setting the switch to *fail-secure mode*. The switches will keep their configuration and unknown packets will be dropped.

4.2.2 Experiments and Results

The primary focus of this subchapter is to see if available COTS switches can provide reliable low latency data transfer services and enforce traffic to guarantee such behavior even in failure cases. Therefore we compare a commercial off-the-shelf Hewlett Packard E3800 switch with a state of the art Rockwell Collins AFDX-3800 switch that is being used in current aircraft such as Airbus A380 and A350. The HP E3800 is running the latest firmware KA.15.16.0008 and offers support for OpenFlow 1.3. In the following we will refer to the switches as COTS switch and AFDX switch.

As OpenFlow controller we use a modified version of NOX supporting OpenFlow 1.3 available on GitHub [116] and the utilities contained within this software. To measure latency and received packet rates an Anritsu MD1230B Ethernet Tester was used. The

device offers network latency measurements with a precision better than $1\mu s$. The rules should be installed with no timeout and the switch's operational mode set to *fail-secure mode*. This way in case of a controller failure, the switch will keep all installed rules and drop all non-matching packets.

To answer the question, whether the COTS switch can offer similar performance than the custom engineered avionic switch, we split the question into three steps. In a first step we evaluate the impact and differences of hardware and software matching of rules on the COTS switch. The instructions will then be extended to apply policing and the accuracy of the policing functionality is checked. In a next step multiple policing meters will be activated and multiple flows matched to emulate multiple flows. In a final step, the switch will be configured with a realistic airplane configuration and directly compared to the custom built AFDX switch. A summary of the evaluation steps are shown in Table 4.2.

#	**Evaluation Goal**	**Means**
#1 #2 #3	Check forwarding latency - normal switching, OpenFlow disabled - OpenFlow and hardware matches - OpenFlow and software matches	An Ethernet testing device is connected as in Figure 4.8 and the forwarding latency is measured.
#4 #5 #6	Check throughput performance - OpenFlow, hardware matches, no meter - OpenFlow, software matches, no meter - OpenFlow, hardware matches, max meters	An Ethernet testing device is connected as in Figure 4.8 and the throughput performance is measured.
#7	Check capabilities of full aircraft config - with all ports and full load	An Ethernet testing device, a load generator and a distribution switch are connected as shown in Figure 4.14.

Table 4.2: Evaluation goals and testing means

All measurements were done with the COTS switch set to 100 Mbps full-duplex mode for comparison to the AFDX switch.

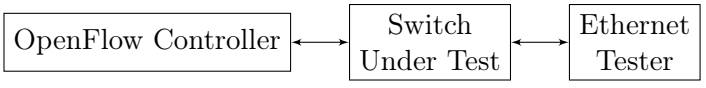


Figure 4.8: Measurement setup used in Section 4.2.2

4.2.2.1 Hardware vs. Software Matching

In a first step we evaluate the COTS switch's capabilities in basic switching. Therefore we instantiate a single matching rule without any meters to get information about the switch's line performance. We then change the amount of packets per second (pps). In

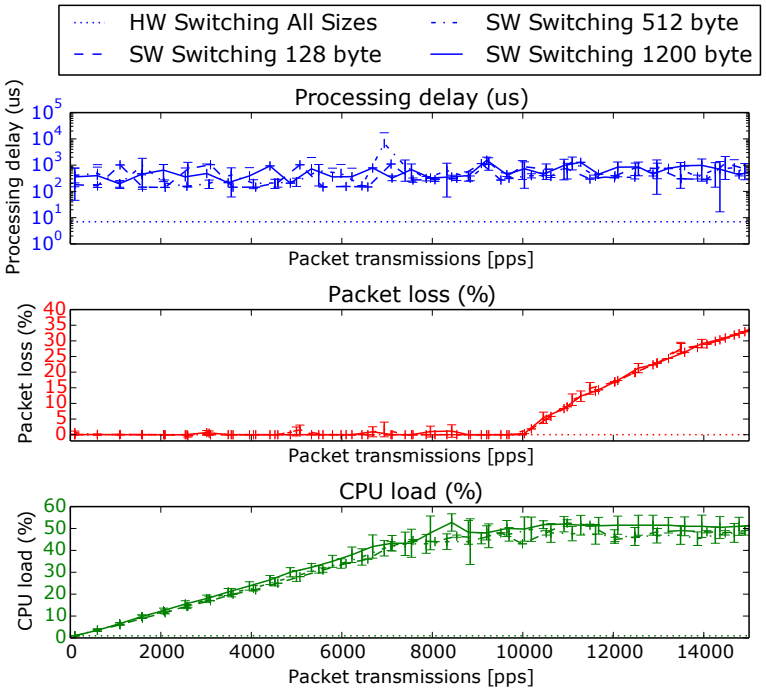


Figure 4.9: Characteristics of forwarding for different packet sizes on COTS switch (errorbars indicate standard deviation of measurements)

Figure 4.9 it can be seen, that the hardware based matching and forwarding works as expected and offers a constant static forwarding delay of around $7\mu s$.

Packets that are switched with software-based matching experience a much higher latency. Those packets have to go through the hardware matching first and be forwarded to the switch's general purpose processor (as shown in Figure 4.10), put through a hash function and then sent back to the switching plane. The experienced latency is around 30 to 100 times higher than for the hardware-matched rules and slightly increasing for higher pps values for the average case. Further the CPU load is slightly bigger for larger packets, which is due to the internal hashing function in the switch that needs to process more payload. Also seen in Figure 4.9 the load linearly increases up to 10.000 pps and then later converges at around 50% processor load.

This convergence is mainly based on the fact that packets are simply dropped above 10,000 pps at the hardware level instead of being forwarded to the software matching function. This seems sensible as the processor is also in charge of running maintenance

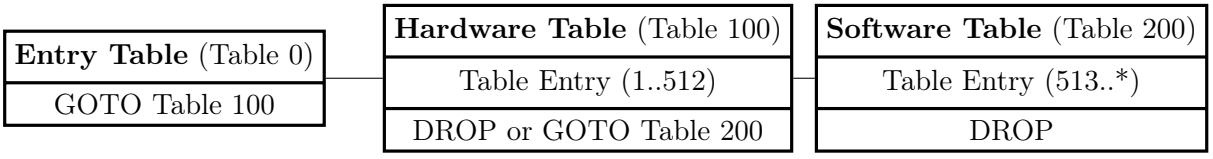


Figure 4.10: OpenFlow 1.3 Table Model and Virtual Link Matching

and management tasks on the switch and without such a limit the switch would be an easy target for denial-of-service attacks. A close-to-zero packet loss can only be observed at speeds of less than 2,000 pps on the whole switch. Between 2,000 and 10,000 pps some sporadic packet losses appear. Such behavior, however, might improve with further firmware updates for the software stack. While the above results are specific to the tested switch, similar behavior is to be expected on all OpenFlow switches.

4.2.2.2 Meter Accuracy

In a next step we evaluate the switch's capability of policing certain flows once matched by a hardware rule. Each meter is realized by a token bucket mechanism that needs configuration of a rate and a burst. In Figure 4.11 it can be seen, that tested rates from 1 to 10,000 Kbps can be policed with the switch. The meter error was calculated according to Equation (4.13).

$$e = \frac{|AllowedRate - ReceivedRate|}{AllowedRate} \tag{4.13}$$

The higher error in the lower meter rates can be explained by the granularity of the measurement using 64-byte packets. The measurement was performed by sending the smallest possible Ethernet frames at full speed and recording of the received rate after policing.

The processor load is constant in idle, which indicates that the metering is certainly done on a hardware level and not passed up to software. The COTS switch that was available

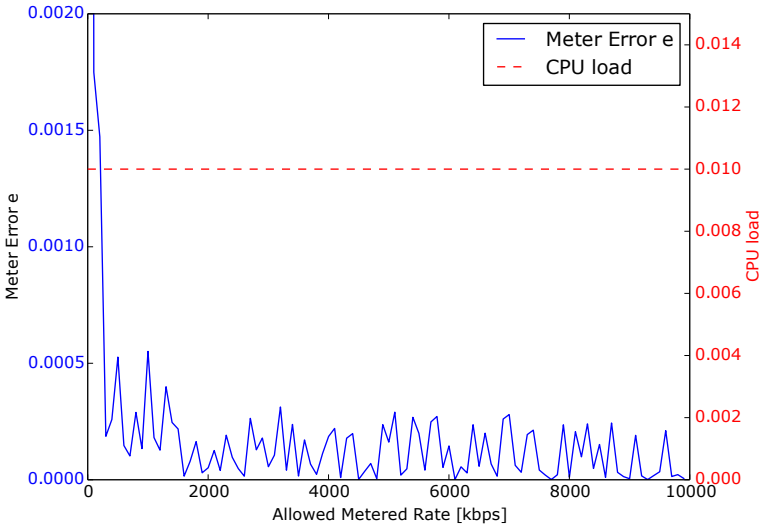


Figure 4.11: Meter accuracy test on COTS switch

allowed us to instantiate around 2,000 meters which will be tested and used in the next section.

4.2.2.3 Multiple Metered Streams

The next section evaluates the switch's capabilities to handle multiple matching rules of flows including the metering of those. The switch we had at hand was not able to map software matched flows to meters. The chart seen in Figure 4.12 is composed of 4 separate measurements of the maximum latency.

The first measurements were done using *normal switching* without any OpenFlow features activated. We see a similar performance as shown before in Figure 4.9. Removing the transmission delay, the switch offers a constant static processing delay of around $7\mu s$. No dependence on the number of flows can be seen, as the switch does not make a difference of the flows here and broadcasts all packets to all ports.

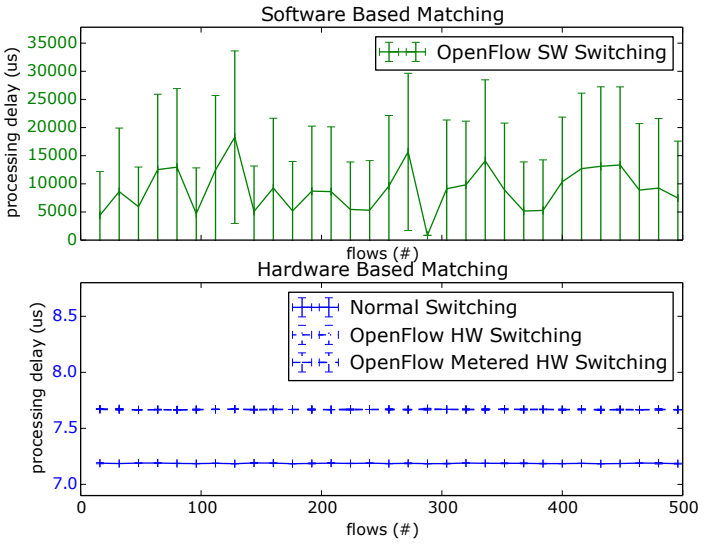


Figure 4.12: Switching latency with multiple parallel flows

The next two measurements were done using OpenFlow hardware matching, switching and metering of the matched flows. The latency again stays constant across the whole number of flows with an OpenFlow incurred delay of around $400ns$ compared to classical switching. No difference is seen between the use of meters and direct forwarding. Furthermore - not depicted here - no difference was seen between forwarding to one output port or several output ports for multicast purposes.

The last measurement in Figure 4.12 concerns the software based matching of packets. The packets were limited to below 10,000pps in total because of the limitations seen in

Section 4.2.2 and only their diversity was increased. The maximum experienced latency is quite unreliable and a factor 30 to 6000 higher than hardware based ones.

4.2.2.4 Failing End-System

A failure affecting an avionic network is, for example, an end-system that gets stuck in a loop and transmits data at a faster rate than expected. This failure is known in the literature as a *babbling idiot*. Babbling idiots are one of the main reasons to apply traffic policing and of major importance to system availability. We evaluate the non-availability of a quality of service enforcement in a real-time system and emulate a crashed end-system that is sending with up to line-speed. It can be seen in Figure 4.13, that such a failure would lead to 100% line load and affects all other traffic as well. Therefore, it would as such break the communication network. The only means to prevent such behavior is to disconnect the failing end-device from the network or block all its traffic.

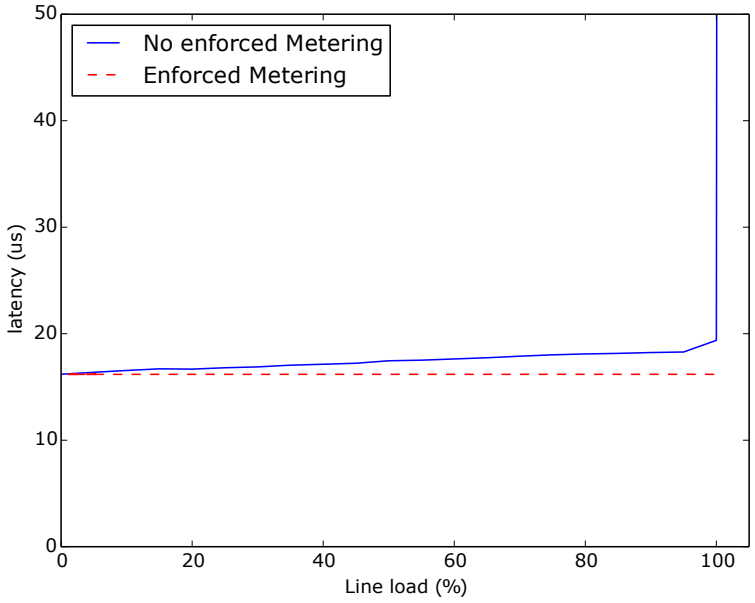


Figure 4.13: Failing end-system comparison

4.2.3 Real-World Scenario

In a last step we took a real aircraft AFDX switch configuration and tried to map it onto the COTS switch. We selected the switch with the highest load in a real aircraft configuration with its key parameters shown in Tables 4.5, 4.3 and 4.4. Due to the limitations seen in the previous sections, only a mapping to the hardware matching was sensible and therefore also giving the limitation of matching up to 512 flows. The

limitation to 512 matches however also meant, that the switch will not be capable to handle a real aircraft configuration yet.

Therefore, we reduced the configuration to the switch's capabilities of 512 flows and ran tests to compare the COTS switch to the real AFDX switch. As shown in Figure 4.14, we used a PC that replayed synthetically created packet capture (PCAP) files. The PCAP files were based on the switch configuration and resembled thus the worst-case load the switch would see in a real aircraft. All virtual links had random starting offsets and multiple seeds were used during the experiments to allow for different phases between the virtual links.

The generated load was fed into a 24 port switch that distributed the load across all its ports according to the desired configuration. The distribution switch was also set-up with OpenFlow for easier configuration. All incoming traffic was then fed back into

Bag (ms)	**Number of VL**
2	20
4	40
8	78
16	142
32	229
64	220
128	255

Table 4.3: AFDX configuration of A380 aircraft (from [46])

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

Table 4.4: AFDX configuration of A380 aircraft (from [46])

Number of VL on switch	649
Number of ports used on switch	19
Worst-case number of incoming packets	33736 pps
Number of VL destination ports	1 to 19
Traffic specification of VL	see Table 4.4

Table 4.5: Realistic parameters used for measurement study

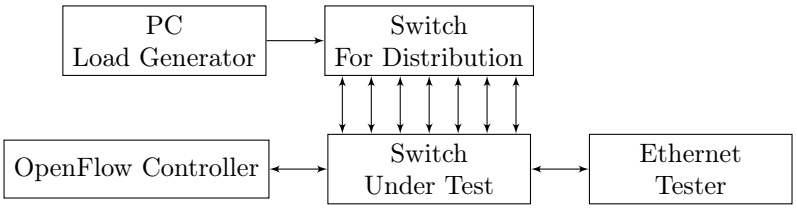


Figure 4.14: Measurement setup with load generator used in Section 4.2.3

the distribution switch and forwarded to the load generator and logged to ensure the functionality of the tested switch.

At the same time an Ethernet Tester was directly connected to the *switch under test* and measured the latency through the switch by sending one test virtual link in and out of the switch.

The results in Figure 4.15 indicate, that the COTS switch is very well capable of handling the reduced configuration with a similar latency characteristic. The AFDX switch has a minimally smaller processing delay of $5\mu s$ compared to $7\mu s$ in the COTS switch. The COTS switch on the other hand has a smaller average latency which might be due to the higher bandwidth in the switching backbone. The maximum values seen on both switches, however, are similar which therefore leaves us to conclude that the COTS switch is indeed able to handle the traffic.

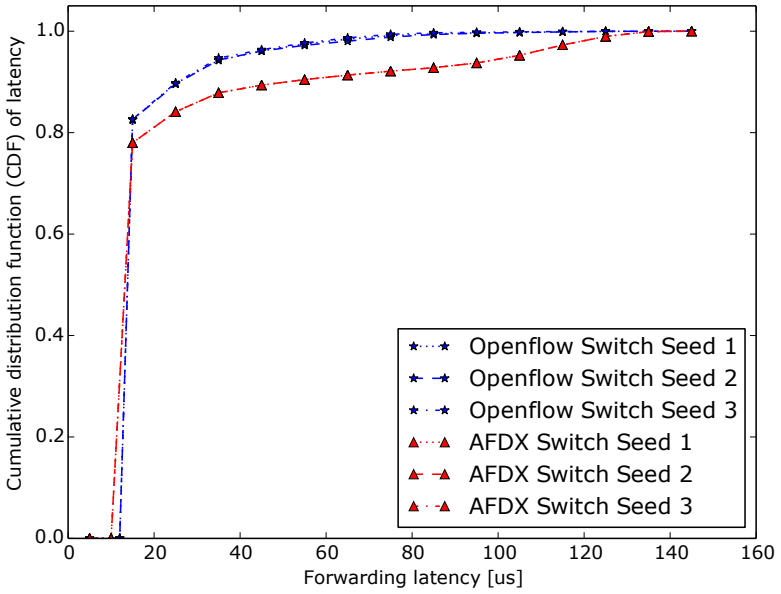


Figure 4.15: Measurements of reduced real-world traffic configuration

4.2.4 Conclusion

This subchapter evaluated the use of COTS switches in an avionic environment. An approach was presented to achieve bounded latencies on standard OpenFlow switches. The approach was explained and a mathematical framework for the calculation of upper bounds was sketched. The solution was then implemented on a COTS switch supporting OpenFlow 1.3 and a measurement study was conducted.

Software based forwarding did not work as expected and results in a factor 30 to 6000 higher latency. However, when hardware based forwarding was used the switch offered good performance, with the limitation of only having 512 entries available. Newer generation switches will likely be able to handle more entries. A promising candidate, that was not available to us at the time of writing, is the Centec V350 switch which according to its data sheet would be able to handle and meter up to 2,000 flows.

As briefly presented in Chapter 3.2, a real-world configuration consists usually of at least 1,000 flows. While the HP switch was not able to handle the full aircraft traffic, in terms of traffic throughput it was able to handle most parts of it and offer similar performance as a state of the art switch that is currently in use in aircraft.

4.3 IEEE Time Sensitive Networking (TSN) with Fault-Tolerance and Frame Preemption

Note This section is part of a publication presented at the 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS) in 2016 [117].

With an increasing amount of solutions for deterministic networks being developed (as presented in Chapter 3.2) no single solution could gain a mass market penetration and thus potentially cheaper prices through high volume sales. Therefore, the standardizing body of Ethernet, the Institute of Electrical and Electronics Engineers (IEEE), decided to start working on a set of networking standards that unifies real-time networks to one common solution that is based on Ethernet, called Time Sensitive Networking (TSN). A mass market potential of TSN is expected due to the interest of the automotive and automation industry alike. If its promise is kept, TSN could be a viable candidate for a future avionic network. Therefore, we look at this collection of standards and evaluate its use for avionics. We give an overview of current state of the standardization, present an open-source network simulation framework developed by us based on which we evaluate TSN in detail and give concluding remarks.

While most parts are still in draft status, the presented simulation framework, called TSimNet, implements the most stable parts of TSN (namely: Qbu, CB and Qci). TSimNet gives other researches a stable base to start evaluating existing networks and future scenarios. An evaluation of frame preemption as well as per-stream policing based on the framework is presented. It is shown, that frame preemption does not always bring latency advantages and highly depends on the configuration of the system.

4.3.1 Overview of IEEE Time Sensitive Networking (TSN)

TSN was designed for a multitude of industrial scenarios, including synchronized zero-jitter use-cases, asynchronous stream use-cases as well as duplicated high dependability use-cases. These use-cases are realized by a multitude of different substandards, each being standardized in their own IEEE project. An overview of all the different substandards and there expected finalization dates can be seen in Table 4.6.

While the name suggests time-awareness of the network, TSN also has a subset of features and mechanisms that can run without time synchronization. If only some parts of all substandards are needed, such use-cases are typically specified in a so-called profile,

naming exactly which features need to be implemented to be compliant with that profile. An example of a profile is AVnu's Ethernet AVB Automotive Certification Profile [118] for IEEE AVB. We envision an industrial TSN profile to include explicit stream forwarding, per-stream filtering, frame replication & recovery and possibly frame preemption.

IEEE Standard	Document Title	Expected Finalization
802.1AS	Timing and Synchronization for Time-Sensitive Applications	Dec 2019
802.1Qbu	Frame Preemption	Published
802.1Qbv	Enhancements for Scheduled Traffic	Published
802.1Qca	Path Control and Reservation	Published
802.1CB	Frame Replication and Elimination for Reliability (Seamless Redundancy)	Review
802.1Qcc	Stream Reservation Protocol (SRP) Enhancements and Performance Improvements	Dec 2017
802.1Qch	Cyclic Queuing and Forwarding	Dec 2019
802.1Qci	Per-Stream Filtering and Policing	Published
802.1CM	Time-Sensitive Networking for Fronthaul	Dec 2019
802.1Qcr	Asynchronous Traffic Shaping	Dec 2020

Table 4.6: Overview of IEEE TSN standards and expected finalization

4.3.1.1 Explicit Stream Identification

The most significant difference between standard Ethernet and TSN is explicit identification of individual streams. Different methods are defined to map packets to streams, all based on either MAC address, VLAN or IP addresses. Based on this stream identifier a TSN-enabled switch will run a different set of separate actions like policing operations, recovery based on sequence numbers and explicit forwarding to one or several output ports. An end-system does not necessarily need to support any additional features. The stream identifier can be explicitly set, or simply matched at the edge port of the switch.

4.3.1.2 Per-Stream Filtering and Policing (PSFP)

Another major feature of TSN is specified in 802.1Qci named Per-Stream Filtering and Policing (PSFP). PSFP allows to police packets on a per stream basis as identified by the stream identification function. When enabled, packets have to undergo certain checks at the device's ingress port, including rate metering, size checks and time-slot compliance checks.

4.3.1.3 Frame Replication and Elimination for Reliability (FRER)

The main idea of FRER, as documented in 802.1CB, is to increase availability of the network by duplication of network packets and delivery through independent paths. To do so, FRER inserts an EtherTag into the packet (see Figure 4.16) indicating the current sequence number of a stream. At the receiving end, such sequence number is recognized at the next stream recovery element and either checked against a bit-vector (*VectorRecoveryAlgorithm*) to allow for out of order delivery or a simple counter that checks, if it is a higher sequence number than its predecessor (*MatchRecoveryAlgorithm*). Inside the relay unit, stream identifiers might get overwritten into a new identifier and by that merged into a single stream. This single merged stream is then subject to cumulative recovery, again based on one of the two previously mentioned recovery algorithms.

4.3.1.4 Frame Premption (Qbu)

Frame preemption, as specified in IEEE 802.1Qbu, allows a network interface to stop sending a packet that is currently transmitting to instead send another higher priority packet. This is done in a way, that the preempted packet is only suspended after at least 64 bytes. The suspended packet does not have to be send again but can resume sending where it left off. For this to work, parts of the Ethernet physical layer had to be modified in a corresponding document 802.3br. The preemption mechanisms can be triggered asynchronously, whenever a high priority packet is arriving, or synchronously to allow jitter free transmission of a pre-scheduled packet.

4.3.2 TSimNet Simulation Framework

In order to evaluate TSN, we implemented a network simulation framework called TSimNet. It has been implemented as an extension of the INET framework [114], which itself relies on the OMNeT++ [113] simulation tool. OMNeT++ is a discrete event-based simulator with a focus on communication networks freely available for academic users. The INET framework extends OMNeT++ in terms of protocols and mechanisms used in current Ethernet networks. Our framework TSimNet extends INET and introduces new features present in TSN on top of it.

We give here implementation details of the main components followed by basic configuration examples. While the focus is on a concise overview, more detailed documentation and the framework itself are available to download at `http://www.eti.uni-siegen.de/es`.

Stream Identification & Encapsulation

To enable TSN-like packet encapsulation, an encapsulation module *TSNEncap* has been introduced that works in a modular way (see Figure 4.16). First off, each frame gets encapsulated in a VLAN tag, which is then used for proper stream identification. In case FRER is enabled on the outgoing link, the packet is then identified and based on the identification, a redundancy tag with a sequence number is added.

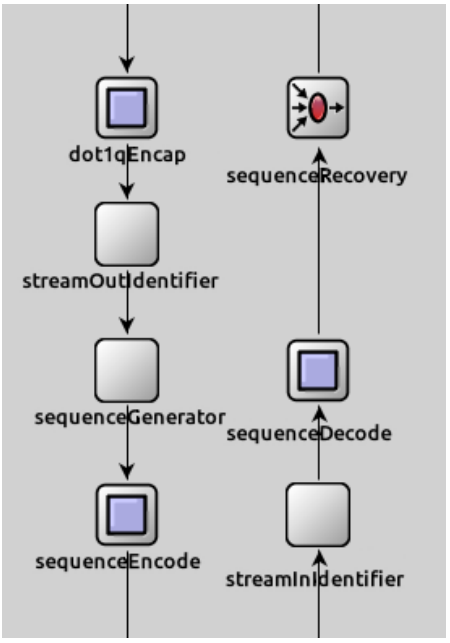


Figure 4.16: TSN Stream Identification and Frame Replication Mechanisms

While stream identification is shown here on the outgoing link, the same component is also used on the incoming link in a switch. The stream identification module identifies the stream based on selectable algorithms. The following mappings are currently implemented: mappings from destination MAC and VLAN tag to one stream identifier, source MAC and VLAN tag to one stream identifier and a mapping based on the packets source IP address and destination IP address.

Stream Forwarding

Another important component is the *TSNRelayUnit* that replaces the standard forwarding unit. Packets arrive through the incoming path of Figure 4.16 into the relay unit. Based on the previous stream identification, packets will be forwarded to one or several outgoing ports or handed to the legacy Ethernet relay unit which does its forwarding based on destination MAC addresses. An illustrative configuration of the *TSNRelayUnit* is seen in Listing 4.1 below. Each entry in line 2 represents the stream identifier and one

or more outgoing ports (e.g.: < stream ident> :< list of ports>). Default dropping for unknown packets can be enabled with lines 3 and 4.

```
1 **.tsnSw.relayUnit.
2       tsn_forwardConfiguration        = "1:1;2:1,2"
3       tsn_dropUnknownTSNTraffic       = true
4       tsn_dropNonTSNTraffic           = false
```

Listing 4.1: Explicit stream forwarding configuration

Stream Filtering and Policing

Inside the MAC layer the *ingressTC* component takes care of traffic policing. While the Qci standard has stream gates in place which are controlled by time-based state machines, all filters here are handled by one stream gate that is always set to open at all times to be independent of time synchronization. Such a stream gate has a filter table that matches each packet to a stream. Each stream entry has to be specified following the MEF 10.3 [119] definitions with Committed Information Rate (CIR), maximum CIR, Committed Burst Size (CBS), Excess Information Rate (EIR), maximum EIR and a dropOnYellow flag to allow color-aware or color-blind traffic filtering. Further a maximum frame size check is configured. All streams not specified in here get forwarded to the relay unit, where unknown traffic can be dropped or broadcasted.

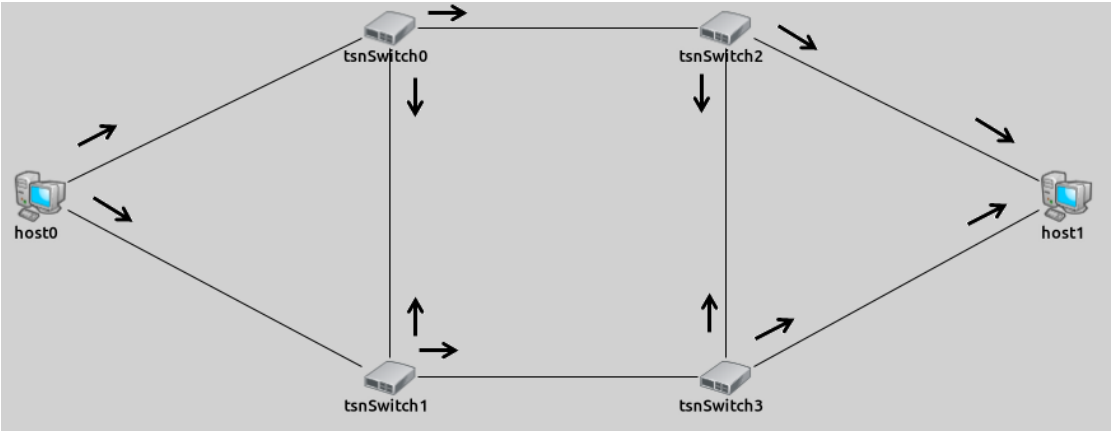


Figure 4.17: Ladder Redundancy as shown in 802.1CB with explicit forwarding

Frame Replication and Elimination for Reliability (FRER)

FRER allows the deduplication of packets with two different kind of mechansisms, a bit vector like mechanism that supports out of order delivery and a simple filter that checks for higher sequence numbers only. Both mechanisms are implemented in the *TSNRelayUnit* and are configurable according to Listing 4.2.

```
**.tsnSeqRecHistoryLength      = 4
**.tsnSeqRecoveryAlgorithm     = "VectorRecovery"
```

Listing 4.2: Frame Replication detailed configuration of *TSNSequenceRecoveryModule*

FRER consists of multiple stages in the relay unit. The first step in FRER is an individual stream recovery with a per-stream filtering at the ingress port before such a stream gets admitted into the ingress handler for policing and later forwarding in the relay unit.

An example explaining this is seen in Figure 4.17, where each switch receives the packet once via each in-port, merges into one single stream and sends out duplicates on each output port again. This way any link can break without affecting the successful delivery of the packet.

```
**.tsnSw.tsn_enableFRERSupport  = true
**.tsnSw.relayUnit.
    frer_individualRecovery     = "2;3"
    frer_mergeConfiguration     = "3:2"
    frer_cumulativeRecovery     = "2"
    tsn_forwardConfiguration    = "2:2,4"
```

Listing 4.3: Frame Replication (FRER) settings in TSimNet's *TSNRelayUnit*

In Listing 4.3 the configuration of a scenario is shown. Each switch is configured with a list of individual recovery streams, a merge list, a cumulative recovery and finally the normal forwarding function of the relay unit to sent out the packet multiple times.

Frame Preemption

Frame preemption can be used to preempt low priority frames in favor of higher priority frames. While in a time-based system such a mechanism could be triggered according to a schedule to make sure the outgoing line is free when the high priority frame arrives, preemption can also be used asynchronously as soon as a new frame arrives. To do so, each stream has to be assigned to a queue in the outgoing port like seen in Listing 4.4.

Stream 1 is assigned to the high priority queue 1 and stream 2 is assigned to a low priority queue. All non TSN traffic will be automatically assigned to the lowest priority queue as configured by the negative value in line 5 in Listing 4.4.

```
1 **.tsnSw.tsn_enableFramePreemption      = true
2 **.tsnSw.eth[0].queue.classifier.
3       tsn_classifyConfiguration         = "1:1;2:2"
4 **.tsnSw.eth[0].tsn_defaultTSNQueue     = 1
5 **.tsnSw.eth[0].tsn_defaultNonTSNQueue  = -1
```

Listing 4.4: Frame preemption configuration

Note that the fragments will be of at least 64 bytes in order to not break compatibility with legacy bridges, therefore, still adding minimal delay to high priority frames.

4.3.2.1 Non-Standard Additions

The following parts are not part of the TSN standards but help with using our simulation framework.

Auto-Configuration Component

The simulation framework requires a lot of parameters to be set which might be error-prone or reduce the usability of the framework. Therefore we extended the configuration component and introduced *TSNConfigurator*. The configurator takes an XML file as input to set the configuration parameters of all nodes, like in *IPv4Configurator* from INET, by introducing a new element: `<tsn-route>`.

For an example see Listing 4.5 where the tsn-route feature is in use. A path element sets the explicit forwarding rules on all switches for this specific stream on its path. Sequence filtering sets individual stream recovery to be applied at each hop of the path while setting traffic parameters for the streaming gates.

This XML file can be used in conjunction with a manual configuration in the scenario files, for duplication of traffic across different paths the tsn-route element can be added for the same stream-id.

```
<config>
  <interface hosts='*' address='10.x.x.x' netmask='255.x.x.x'/>
  <tsn-route stream-id="1"
    src="host0" dst="host1"
    sequence-filtering="enabled"/>
    ....
</config>
```

Listing 4.5: Auto-Configuration config.xml

Failure Injection

In order to test the new mechanisms, faulty devices and hosts have been added. The first faulty module is the *TSNBabblingIdiotHost*, which keeps on sending an arbitrary amount of data to random destinations besides the traffic it is configured to transmit in normal operation. Using this node we can simulate traffic overload in the network and verify functional traffic policing modules and configuration settings.

The second faulty component is the *TSNRandomSequenceGenerator*, which is also included in the aforementioned babbling idiot node. The random sequence generator injects randomly increasing sequence numbers into packets of a particular stream, therefore bypassing the recovery functions of the nodes on the way. In an ill-configured system such duplicates would then be forwarded normally by the relay unit possibly congesting other parts of the network. Such a fault has to be recovered through traffic policing.

4.3.2.2 Computational Overhead

To check our framework's performance, we checked the overhead of the framework's new functionality in comparison to the INET framework. Table 4.7 gives an overview of added events, added CPU time and added memory consumption for the scenario used in Section 4.3.3.1 with three nodes and one central switch.

Basic TSimNet represents a scenario, where only stream encapsulation, stream identification and explicit forwarding are enabled. Due to the graphical encapsulation used (see Figure 4.16), events are scheduled for each subcomponent in the encapsulation module explaining the increase in events. Similar to this also CPU time increased in this scenario to handle all new events. The number of added events will remain a constant factor with increasing numbers of packets.

Traffic policing added a limited number of events, incurred through the extra components in the MAC. Also CPU time increased in order to do the filter look-up and apply

policing to the flows. Recovery (FRER) features didn't add any extra events, however, needed extra CPU time to run the recovery algorithms and look-up of sequence history of flows. Frame preemption added extra events, as the fragmented packets get scheduled and retransmitted through the transmission channels. CPU time increased because of that, also memory consumption did go up due to the higher number of packets in the simulation.

With all features enabled, about 70 percent extra events have to be scheduled for the TSimNet enabled simulation framework, due to the modular nature of some of TSimNet's features. This value could be reduced at the expense of less readability and no graphical references to the standard documents. Memory consumption stays roughly the same for this simulation as the selected scenario was a small one.

	standard INET	basic TSimNet	policing TSimNet	recovery TSimNet
# events [ev∗10^3]	3799	5401 (+42%)	5601 (+47%)	5401 (+42%)
CPU time [sec]	20,1	35,54 (+76%)	36,55 (+81%)	36,04 (+79%)
memory [Mbyte]	69	69 (+0%)	69 (+0%)	69 (+0%)

	standard INET	preemption TSimNet	all features TSimNet
# events [ev∗10^3]	3799	6268 (+64%)	6468 (+70%)
CPU time [sec]	20,1	39,28 (+95%)	39,72 (+97%)
memory [Mbyte]	69	70 (+1%)	70 (+1%)

Table 4.7: Performance evaluation of TSimNet simulation framework in comparison with standard INET (Times indicate averages of the runs)

Concluding the frameworks performance, it can be seen, that only CPU utilization increases but memory consumption stays mostly constant. This is due to the fact that no states, except the sequence numbers, are kept in the devices to allow for an efficient implementation in ASIC devices.

4.3.3 Evaluation

Building on our framework we present a first performance evaluation of TSN with a focus on our use-case. The evaluation was done with objectives listed in Table 4.8 and a focus on non-synchronized systems to make use of the verification techniques in Chapter 2.4.

#	**Evaluation Goal**	**Means**
#1 #2	Evaluate benefits of frame preemption - in a simple one node use-case - in an industrial use-case	Run simulation software with different scenarios (4.3.3.1 and 4.3.3.2) to determine worst cases.
#3 #4	Ensure similar or greater performance compared to legacy solutions - checking compatibility to legacy features - ensure sufficient resources available	Define a TSN profile that states requirements of all sub-standards used (4.3.3.3).

Table 4.8: Evaluation goals and testing means

4.3.3.1 Simple Line Congestion

A first evaluation is looking at the frame preemption capabilities of TSN. We use a simple topology consisting of three nodes like seen in Figure 4.18, where one node is sending important control traffic to a sink and at the same time a best-effort node is congesting the network with cross-traffic with increasing lineload. Traffic is generated by using random sized packets that transmit at random times, limited by a token bucket shaper that has its token rate set to the indicated lineload.

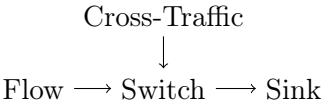


Figure 4.18: Set-up used for simple line congestion evaluation

Three scenarios are shown in Figure 4.19, each annotated with its minimum, average and maximum values. The first scenario with no prioritization and no preemption shows typical congestion results with high latencies especially at high load factors. Once prioritization comes into effect, the worst case latency stays lower especially at high load factors. This is due to the fact, that at most one best-effort packet can block the priority packet. In the third scenario we apply different priorities combined with frame preemption. It can be seen, that frame preemption brings down latency for the maximum case and especially on the average case performs well with close to no correlation to the lineload.

We conclude that frame preemption does indeed offer latency advantages over non-preempting scenarios, however, only for one packet at the cost of adding extra complexity to sender and receiver.

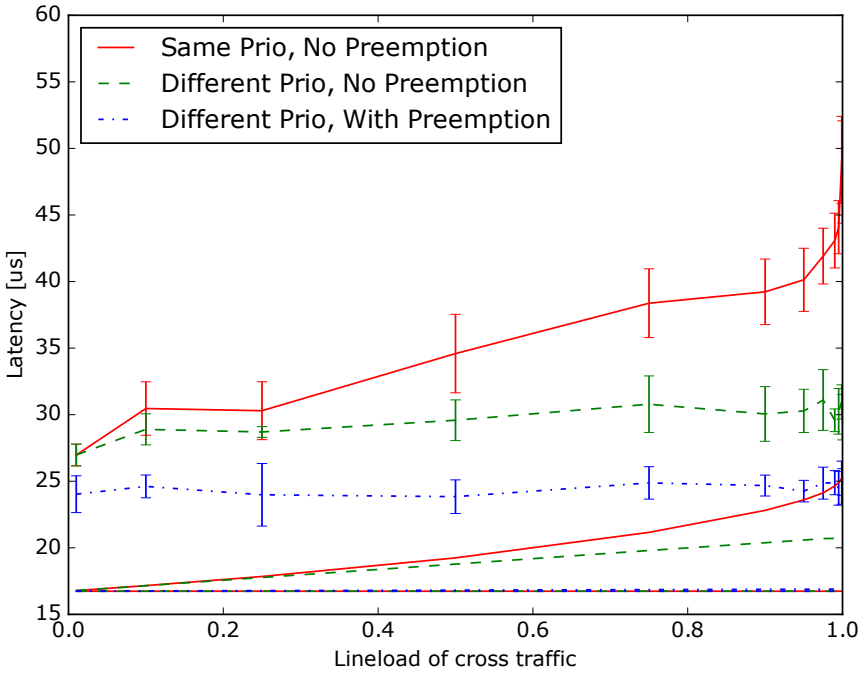


Figure 4.19: Comparison of min, max and avg end-to-end latencies with and without Frame Preemption (setup: two nodes; line speed 1Gbps; error bar indicates std deviation)

4.3.3.2 Industrial Line-Topology Preemption

In a next evaluation we investigate the use of frame preemption in typical industrial line-topology settings. In such a setting, individual devices are usually computationally weak and try to minimize extra efforts, therefore backbone (or in our case ring) traffic is typically scheduled first in order to minimize buffering at the local node. Resembling this, we assume a scenario like seen in Figure 4.20 for further evaluation, where a flow has to go through three nodes before reaching its destination sink and calculate its worst-case end-to-end latency.

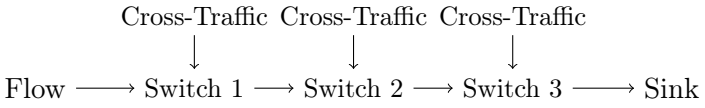


Figure 4.20: Set-up used for industrial line-topology evaluation

In Figure 4.21 such latencies for the described path are depicted. The first column represents the latency in an idle store-and-forward scenario, where all queues are free and only transmission delay is taken into account across the switches with a zero processing delay. The second column then represents the worst case option while having no frame preemption enabled, however, prioritization in the outgoing queue, e.g. ring-traffic first.

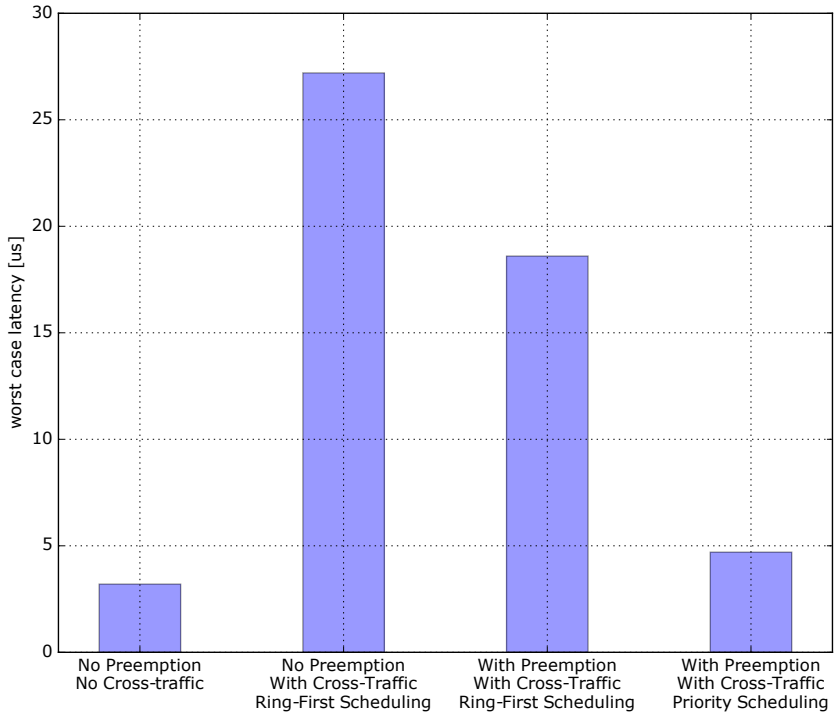


Figure 4.21: Comparison of maximum end-to-end latencies in an industrial line scenario with and without frame Preemption (setup: three switches; line speed 1Gbps; control packet size 100B; cross traffic packet size 1000B)

At each hop, a maximum size Ethernet frame might just start transmitting from the local node so that the high priority frame has to queue until the transmission is finished, therefore giving a queuing time of one large cross traffic frame at each switch.

Column 3 represents the same scenario as column 2, however, with frame preemption enabled and ring-first scheduling. The value is significantly larger than column 4, where the high priority frame is always prioritized, while in column 3 the high priority frame might be blocked. If the last part of an already preempted frame is in front of a high priority frame, the reassembled frame will start transmitting from the next ring node and cannot be preempted as stacked preemption is not allowed for in the standard. Therefore, if frame preemption is to be used in a backbone-first scenario the latency advantages are not as significant as they could be in a properly configured TSN network.

4.3.3.3 Industrial TSN Profile

We propose here a profile that will be sufficient to fulfill the function of current avionic networks and with self-configuration covering future scenarios. An example of a profile is AVnu's Ethernet AVB Automotive Certification Profile [118] for IEEE AVB.

We envision an industrial TSN profile to include explicit stream forwarding, per-stream filtering, frame replication & recovery as defined below in Profile 1.

Basic Protocol Support
The following protocols must be supported in its minimal forms as defined within the standards: **Switches**: 802.1CB - For explicit stream forwarding in switches 802.1CB - For packet duplication and deduplication 802.1Qci - For per-stream policing of streams **End-Systems**: 802.1CB - For explicit stream forwarding in switches
Advanced 802.1CB Stream Identification Options
All devices need to support the source MAC + VLAN stream identification function. At least 2^{16} unique mappings need to exist.
Advanced 802.1CB Duplication Options
Switches: All switches need to support the *MatchRecoveryAlgorithm* for at least 8192 unique streams.
Advanced 802.1Qci Options
Switches: All switches need to be able to police up to 16384 unique streams on a configurable per-stream basis.

Profile 1: Industrial/Avionic TSN Profile

4.3.4 Conclusion

In this subchapter we presented a TSN simulation framework for OMNeT++ that implements all features needed for an industrial profile that is not based on time synchronization. An evaluation of frame preemption as well as per-stream policing based on the framework were presented and it was shown, that frame preemption does not always bring significant latency advantages and highly depends on configuration of the system. The full source code as well as documentation is available via www.eti.uni-siegen.de. We hope this framework gives means to developers to evaluate new TSN features by open-sourcing this code.

4.4 Results & Evaluation

In this chapter we presented three concepts and mechanisms based on the aeronautical requirements from Chapter 3.1. Each concept aimed to fill gaps found in current state of the art literature, to push the area of communication networks forward. An overview of each subchapter's contribution is summarized in Table 4.9.

Real-Time Network	Chapter	Support									
		Traffic			Configuration			Fault			
		Time-Triggered	Rate-Constrained	Best Effort	In-Band Configuration	Online Reconfiguration	Vendor Independent	Fault Detection	Fault Isolation	Fault Masking	Suitability for FAN[1]
HSR	3.2	**n**	**n**	y	y	**n**	n	y	n	y	o
Heise et al. [111]	4.1	**y**	**y**	y	y	**y**	n	y	n	y	o
OpenFlow	3.2	n	**n**	y	n	y	y	y	**n**	n	+
Heise et al. [115]	4.2	n	**y**	y	n	y	y	y	**y**	n	+
IEEE TSN	3.2	y[2]	y[2]	y[2]	n[2]	y[2]	y[2]	y[2]	y[2]	y[2]	**o**
Heise et al. [117]	4.3	y[2]	y[2]	y[2]	n[2]	y[2]	y[2]	y[2]	y[2]	y[2]	+

Table 4.9: Contributions of this Chapter

4.4.1 Evaluation

In Chapter 4.1 a mechanism to guarantee timely packet arrival in HSR was developed and implemented in a network simulation. Further, a mathematical model for the mechanism has been developed and compared with the simulation results. It was shown, that time-awareness offers better latency in higher load scenarios. However, it was also shown that the loss of time-awareness increases the latency significantly and it has to be carefully evaluated, if such a mechanism is to be used in aeronautics.

The next Chapter 4.2 evaluated the use of COTS switches in an avionic environment. An approach was presented to achieve bounded latencies on standard OpenFlow switches. The approach was explained and a mathematical framework for calculation of upper bounds given. It was then implemented on a COTS switch supporting OpenFlow 1.3

[1] Future Avionic Network
[2] Features expected once standardized

and a measurement study was conducted. Software based forwarding did not work as expected and results in a factor 30 to 6000 higher latency. However, when hardware based forwarding was used the switch offered good performance and could offer similar performance as a state of the art switch that is currently in use in aircraft.

In the last Chapter 4.3 we presented a TSN simulation framework for OMNeT++ that implemented all features needed for an industrial profile that is not based on time synchronization. An evaluation of frame preemption as well as per-stream policing based on the framework were presented and it was shown, that frame preemption does not always bring latency advantages and highly depends on the configuration of the system.

4.4.2 Interpretation

In this section we interpret the results achieved in this chapter and put them into context of a future avionic network.

Chapter 4.1 offers rate-constrained and a time-based mechanism without the need for global time synchronization. While this improves the traffic handling, one major flaw of HSR has not been fixed that is the MAC address collision problem within the deduplication table in HSR. As the standard dictates a hash function, it cannot be guaranteed that collisions in the table will not occur and as such deduplication for packets might be flawed. As plug and play is a major selling point of HSR and the mechanisms were implemented with ease of use in mind, this flaw cannot easily be fixed.

In Chapter 4.2 OpenFlow was made deterministic and it was shown to offer similar performance as current switches while enforcing traffic correctness. By doing so we also added fault isolation as messages of misbehaving devices will be stopped every hop along the way. The benefit of this is already a big saver in cost and can offer great advantages over current systems. A downside of OpenFlow is the out of band network, which is addressed in Chapter 5.3.

While IEEE TSN in Chapter 4.3 aims to support the same features as deterministic OpenFlow and more, it is still being standardized and offers too many sub standards that will not likely be embedded in one network chip. With the proposed aeronautical profile we show a means forward to allow IEEE TSN in the aeronautical community to potentially offer cost savings and all functionality needed without a proprietary solution.

An integration of OpenFlow's ideas of remote configuration is seen with TSN's YANG. The YANG working groups in IEEE try to standardize all of TSN's configuration parameters to have remotely configurable switches, specifically in the areas of explicit path

configuration. This is still an area of active standardization and ongoing at the time of writing this thesis.

Chapter 5

Self-Configuring Network

One of the major tasks when deploying real-time networks as presented in Chapter 4 is their configuration to achieve real-time behavior. In this chapter we present mechanisms for self-configuring plug and play real-time networks that automatically set up devices and offer hard real-time guarantees to such devices based on the underlying network used. A new architecture is proposed to achieve a system that can react to failures of switches and links by trying to repair such failures on a network level and restoring a full redundancy level while maintaining hard real-time guarantees. It is shown, that the solution can achieve a seamless failure recovery from link failures without any complexity at the end-device side. Further, we present a way to achieve bounded configuration times, by introducing a safe in-band channel that enforces traffic characteristics on the in-band control message while getting rid of the out-of-band connection.

In this chapter we present our work that aims to close gaps found in Chapter 3, like support for real-time networks, online reconfiguration and guarantees that configuration traffic will not interfere with operational traffic. The work has been presented at various scientific conferences, as indicated at the beginning of each sub-chapter.

Structure of this chapter

In Section 5.1 we start this chapter with an introduction to the process of network configuration for civil aircraft and give an overview over the configured parameters. We then present a self-configuring plug and play network in Section 5.2, that relies on OpenFlow as network service and supports easy integration of new devices through a subscription protocol. In Section 5.3 a safe in-band configuration channel is presented to allow configuration of switches without the need for external cabling like usually done in

OpenFlow. Finally, we compare results and give an evaluation of the solutions presented here in Section 5.4.

5.1 System Model

In this section we describe the overall configuration process of network switches in a typical commercial aircraft setting. Figure 5.1 gives an overview of all steps in the process. Most importantly it has to be noted, that switch configuration is performed at design time and not changed later. Multiple configurations per aircraft are generated to allow some flexibility.

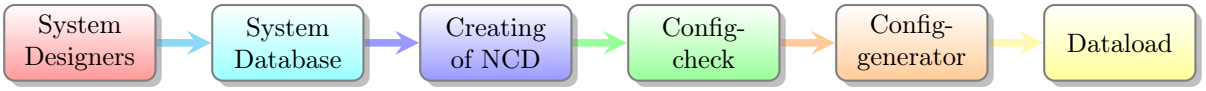


Figure 5.1: Overview of Network Configuration Process (via [4])

1. **System Designers** collect the communication needs for the subsystem they are designing and its delay requirements.
2. A **System Database** then collects all communication requirements of all individual subsystems.
3. Next a **Network Configuration Description (NCD)** file is created based on the needs for a specific aircraft with all dependencies resolved.
4. The NCD file is then used as input to the **config check tool** which performs formal verification to ensure latency, jitter and buffer requirements are provably fulfilled.
5. Afterwards the verified file is **translated** into a machine readable format and archived with a **part number** before being loaded onto the airplane.

5.1.1 Configurable Parameters

Figure 5.2 gives an overview of three types of devices typically seen in a cabin network: one generic network device and two specific end-devices. In such a network three types of parameters can be distinguished: predefined, self-configured and non self-configurable parameters.

Predefined or Static Parameters

Such parameters do not have to be configured during run-time. An example of this is the unique L2 MAC address for each device. If no such address is available, a mechanism

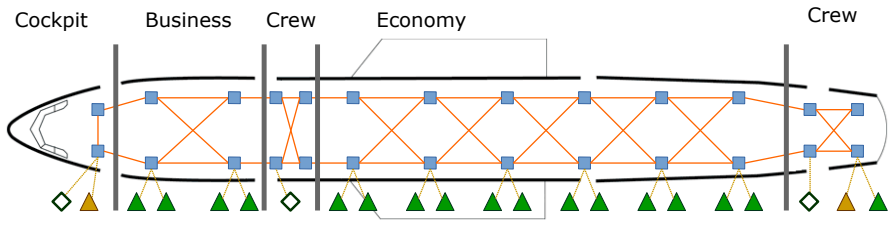


Figure 5.2: Meshed cabin network with different zones - ♢ representing sources, △ sinks and □ switches

like rollout/rollin procedure [110] can be used to assign it. Furthermore, a device needs to have local knowledge about its device class as well as the services offered by it in terms of network calculus describable curves including traffic characteristic, maximum latency, jitter and reliability requirements. The particular instance of each device can be auto-configured.

Self-Configured or Learned

Automatically learned parameters include the topology of the backbone network architecture as well as the location of end-systems. Further, such end-systems should be automatically matched between a service agent that is offering a service and a user requesting a service. Once a service is requested, the route shall automatically be found in the network and set-up according to the requirements specified by the device.

Non self-configurable

The third type of parameters are those, that cannot be auto-learned. An example here is device instance specific information, for example if a speaker is installed in economy or business class and the suitable subscription to audio channels. Such information has to be either preinstalled in the software, either in the device through an ad-hoc channel or some central control unit that has a central configuration file. Also, all information derived from those device specific instances, for example priority of travel classes over other travel classes, like seen in Listing 5.1, cannot be learned.

5.2 Self-Configuring Plug and Play Real-Time Network

Note This section is part of a publication presented at 22nd IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN) in 2016 [120].

In this subchapter we present an approach for a self-configuring plug and play network that automatically sets up devices and offers hard real-time guarantees to such devices. A new architecture and a protocol based on OpenFlow (see Chapter 3.2.2.3) are proposed to achieve a system that can react to failures of switches and links by trying to repair such failures on a network level and restoring a full redundancy level while maintaining hard real-time guarantees. Implementation details are explained and the architecture is evaluated against randomly generated topologies. It is shown, that the solution can achieve a seamless failure recovery in case of link failures without any complexity at the end-device side.

5.2.1 Concept & System Description

To build a network that offers the features needed for a future avionic network, four main building blocks are needed: (A) awareness of the topology in the network as well as on the end systems, (B) service discovery within that network and fallback options for COTS devices that don't implement the service discovery protocol, (C) quality of service mechanisms guaranteeing latency once the communication endpoints are known and (D) reaction to failures in the system. Each of the points are described in the following.

5.2.1.1 Network Control and Topology Awareness

Protocols like *Spanning Tree Protocol* (STP) or *IS-IS PCR* are typically used to achieve loop-free communication paths. Another way is the use of OpenFlow, where the controller has a central view and connections to all switches and thus can redirect all unknown or new packets to itself. Assuming an out-of-band network, all devices are about one hop away from the central controller. Therefore, a direct neighbor discovery protocol is sufficient. This mechanism implemented in an OpenFlow controller is called OpenFlow Discovery Protocol (OFDP) [96]. The controller generates LLDP packets and lets the switch forward them to its ports. Received LLDP packets are forwarded back to the controller by an OpenFlow rule installed beforehand on the switches. Additionally, OpenFlow adds meta-information to the packet like the receiving port. Using OFDP

the controller can then build a topology view in the controller and find loop-free routes. Combining this with OpenFlow link events results in a live topology view.

5.2.1.2 Service Discovery

The same mechanism can be reused on a switch in order to forward any packets of attached end-devices to the controller. This way, we are able to reuse this mechanism not only for detection of a presence but also as means for a logically centralized service discovery. Service discovery is typically divided into structured, unstructured and hybrid architectures. Unstructured architectures do not have a central node and therefore need support for partial-matching thus adding complexity. Hybrid architectures often make use of clustering by introducing a loose structure like an overlay. Structured architectures store their knowledge on one or more fixed nodes that need to answer all queries. As these nodes have the full directory, the service search time becomes predictable. Due to the centralized nature of OpenFlow the structured architecture fits best here.

One common implementation of the structured centralized approach is Service Location Protocol (SLP) [121]. In SLP three basic roles are used: *user agents* (UA), *service agents* (SA) and *directory agents* (DA). While a UA is a client that sends service requests, the SA on the other hand provides services. The optional DA brings both together. A typical overview can be seen in Figure 5.3. For this approach to work, the end-systems need to support UA and SA requests. The DA will be placed in a central node, like the OpenFlow controller.

We make use of the SLP protocol with its *service requests* for UA and *service registrations* for SA. Additionally, we introduce an extension, the grouping feature, to allow for dynamic creation of multicast groups. A typical request now looks like Listing 5.1.

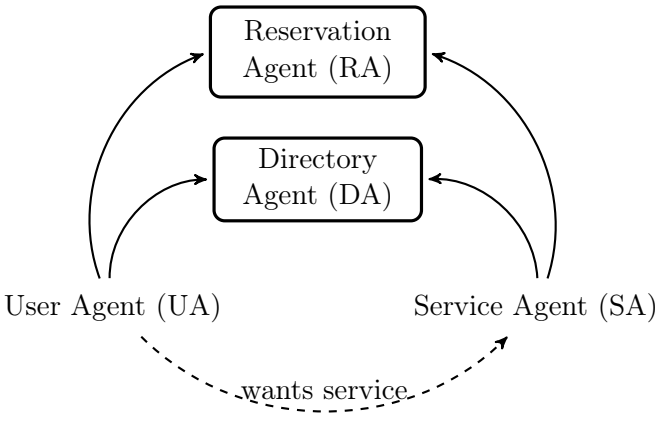


Figure 5.3: Centralized approach of extended SLP

```
service:speaker:rtp://1.2.3.4
  (location-class=economy), (location-row=12), (allow-grouping=1)
```

Listing 5.1: Service registration with our grouping extension

When the *allow-grouping* flag is set, the service allows the directory agent to transparently put the device into a multicast group with other devices fulfilling the attributes of a search request. To find a service, the UA will send a service request towards the DA and the DA will answer with either a device, a multicast group transparent for the UA or an empty message.

```
1  function msgRegReqHandler(msg ∈ {REG, REQ}):
2    %% Save registrations and requests.
3    if msg = REG:
4      RegList ← RegList ∪ {msg}
5    if msg = REQ:
6      ReqList ← ReqList ∪ {msg}
7    %% Get requests that are affected
8    if msg = REG:
9      for req in (msg matching ReqList):
10       curReqList ← curReqList ∪ {req}
11   if msg = REQ:
12     curReqList ← {msg}
13   %% Send replies to requesting users
14   for curReq in curReqList:
15     matchSet ← curReq matching RegList
16     %% This also creates new multicast groups
17     matchSet ← matchSet reduced by groups
18     for match in matchSet:
19       if match = Grouped AND not in MulticastList:
20         MulticastList ← MulticastList ∪ {match}
21       match ← MulticastList[match]
22       send reply to match->src
23
```

Listing 5.2: Handling of register and request messages

The DA, however, will always remember the UA's request and send a DAAdvert message containing a service, once the service is available. The pseudo code used for finding multicast groups is explained in Listing 5.2.

Lines 1-7 store all incoming registrations and requests. As a registration might match to more than one request, lines 10-14 calculate the affected requests by such an incoming message and returns a list. Line 18 calculates the registrations matched by each request and reduces the result set to groups. The remaining lines then save such groups and inform the UA about changes to the request.

```
reserve:speaker:rtp://1.2.3.4
  (traffic-packet_size=100),
  (traffic-packet_interval=100ms),
  (traffic-max_delay=100ms),
  (traffic-reliability_factor=2)
```

Listing 5.3: Service reservation

After having received the grouped entries, the UA selects a service based on the SLP service URL and sends a reservation request to the reservation agent (RA). A typical reservation can be seen in Listing 5.3. Besides the URL, this request contains a traffic specification that the UA is going to transmit towards the service, namely a maximum latency that has to be fulfilled.

Further a reliability factor is introduced, which represents the number of disjunctive paths the RA is supposed to reserve. Once reserved, the UA can start normal operations.

Service discovery for COTS devices

Devices that are not able to send registration messages can be integrated by the discovery via their distinct features. A packet sent out by an unknown device will be forwarded to the controller through OpenFlow's packet-in method. The controller is then able to classify the device by some distinctive feature, for example the device's MAC address, and send a registration request to the DA. The device is then registered in the network and discoverable by service-discovery supporting devices.

5.2.1.3 Constraining Traffic

At reservation time, the RA is queried to calculate one or more routes between the end-points. Users that request a service send a traffic description, containing rate and maximum packet size, a quality request containing maximum end-to-end latency as well as a reliability factor. All traffic descriptions are collected and network calculus is applied to calculate the upper latency limits for each stream. Such results are compared with the quality requests in the registration and if all existing and new reservations are satisfied, the reservation request is accepted and installed in the network. During the run time of the system new services will be offered or removed. As such, only the additional flows need to be scheduled and only intersecting flows rechecked, which leads to a decrease in computation time. For simplicity purposes we used a simple list scheduling approach with priorities on the time of registration. However, for further work on incremental scheduling we refer to other works in the field like [122]. A shortest path algorithm was

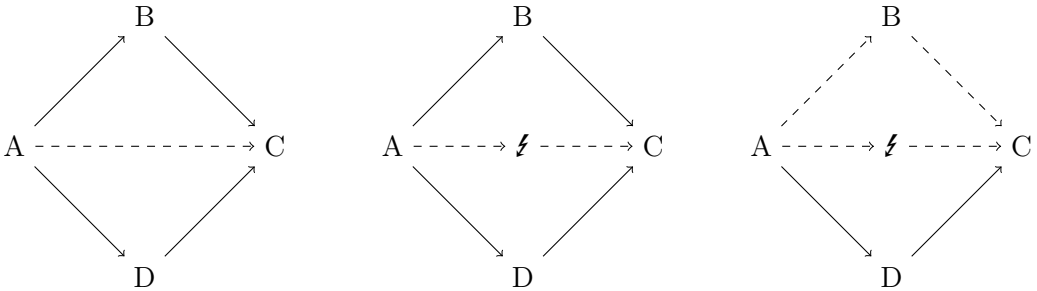
used to compute the routes. For redundant flows, such a route was then removed from the network and the algorithm run again.

To achieve determinism and enforce correct traffic behavior we build on our works of Chapter 4.2, that applies ingress traffic policing by using native OpenFlow meter instructions. If a packet can't be mapped to a flow in (1) or exceeds the rate allowed in (2), the packet will be dropped before entering the switch's backplane. This way only conformant packets are allowed in and with knowledge about the switch's line speed the maximum queuing time and buffer size can be calculated by using network calculus as presented in Chapter 2.4.1.

5.2.1.4 Reaction to Failures

In order to react to failures in the network, the system has to be aware of the health of each of its parts. A failure of the controller can be circumvented by running multiple controllers in synchronization with one in hot standby. Due to OpenFlow's report capabilities, every link event generates a message in the switch which forwards it to the controller instantly. In the event of a cable failure with a link going down, a new route can be computed if a backup path is theoretically available.

All requests and registrations from Section 5.2.1.2 are stored and are recomputed upon such an event. During reservation, a reliability factor was transmitted which decides on the failure handling according to Figure 5.4. In case of a reliability factor of two or above, packets will be duplicated in the network and routed multiple times toward the destination. While deduplication is not possible with OpenFlow, we point here towards other technologies like IEEE's TSN 802.1 CB [1], where such a feature will be available. In case of a link failure concerning a redundant flow, the flow on the failed link will



(a) Working Case

(b) Failure; Reliability factor 1; fail-over time $\geq$ 0ms

(c) Failure; Reliability factor 2; fail-over time 0ms

Figure 5.4: Illustration of reliability factor in failure case (arrows indicate links, dashed arrows links with an active flow reservation)

be rerouted while the working route remains untouched. Therefore, no packets are lost between the end-points.

5.2.2 Implementation

The proposed concept has been implemented for its evaluation. As OpenFlow controller the Ryu controller [123] has been selected due to its easy extendability and open source license. Two switches of different vendors were connected to the controller as well as the Mininet emulator with a connection to the real switches to allow seamless testing between virtual as well as real environments with both being connected to the same controller at the same time. While the switches implement OpenFlow meters, Mininet only offers a stub function for this and thus does not do any actual traffic policing.

The Ryu controller offers built-in OFDP topology discovery and was extended by the service discovery module as well as a connection to the DiscoDNC [44] framework, which was used to calculate latency bounds for each installed flow at the time of a request.

5.2.3 Discussion & Evaluation

To evaluate and test the proposed architecture we used different randomly generated topologies. Different topologies are generated and a full system boot-up time is measured as well as the time for failure recovery. The measurements were done in Mininet [124] and are expected to be similar to real switches.

The topologies generated during evaluation were chosen with a focus on survivability, with a backup path always physically available. This way the controller has means to reroute the traffic and heal the network. In terms of graph theory, this is expressed as the k-connectivity property of a graph, indicating how many links can be removed from the graph before the network is split into two parts. We therefore need networks with a k-connectivity of more than 2.

Parameter	Value
Number of nodes	{50, 80}
Number of switches	{50, 80}
Host boot delay	uniform random(1s,5s)
Host destination pair	random(all nodes)
Network k-connectivity (Outgoing links)	≥ 2
Networks generated	100

Table 5.1: Simulation parameters for evaluation

The parameters chosen for the measurement study can be seen in Table 5.1. Nodes have a boot delay of up to 5 seconds, where they do not communicate and only start the service discovery algorithm after this time. Each experiment was conducted 100 times with different seeds and thus topologies. Destinations were picked randomly, in case of multiple subscriptions to the same destination, multicast was applied. Rules and meters were then generated and installed with OpenFlow in the network.

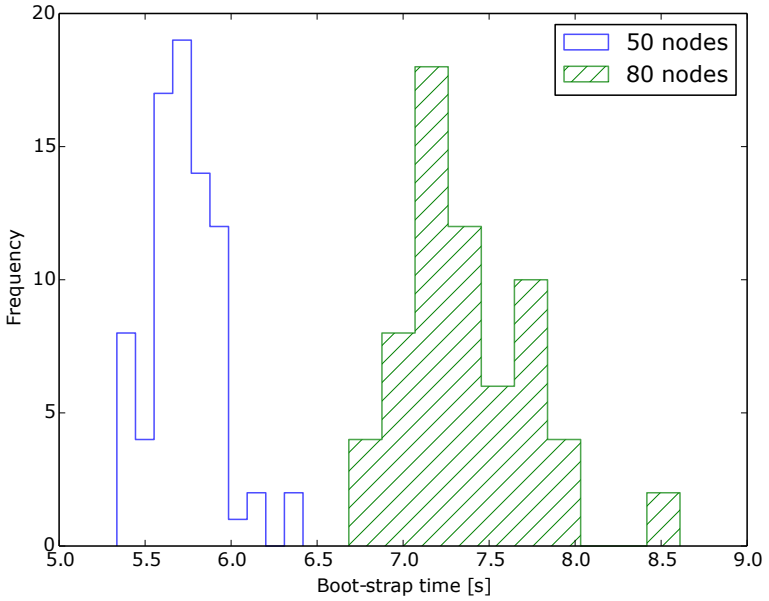


Figure 5.5: Experiment 1: Network Boot-Strap time

Experiment 1 measured the time of the network from cold start till all nodes are fully discovered and paths configured. A configured path consists of a rule for matching a destination MAC address to an output port, as well as a meter on each switch along the path. The meters will police such a path according to the service requests traffic description. The experiment results can be seen in Figure 5.5. Runs with 50 and 80 nodes were done. For both runs a uniform distribution is visible. A longer configuration time is seen for 80 nodes, as it is more likely that the last node will only boot closer to the 5 second mark, as well as the increased flow count that has to be placed in the network. Higher values on both runs are seen with networks that have more links, which increases workload for the path finding algorithm.

In a second experiment we investigated how long it takes for the network to recover after a switch has failed. In case the traffic is duplicated on a network level, this represents the time till the full redundancy level is restored. The results are shown in Figure 5.6. The main influencing factor here is the number of affected flows, that have to be rerouted. The time from the event occurrence till an event at the controller is in the range of milliseconds. Failure recovery for more nodes takes more time as the paths are on average

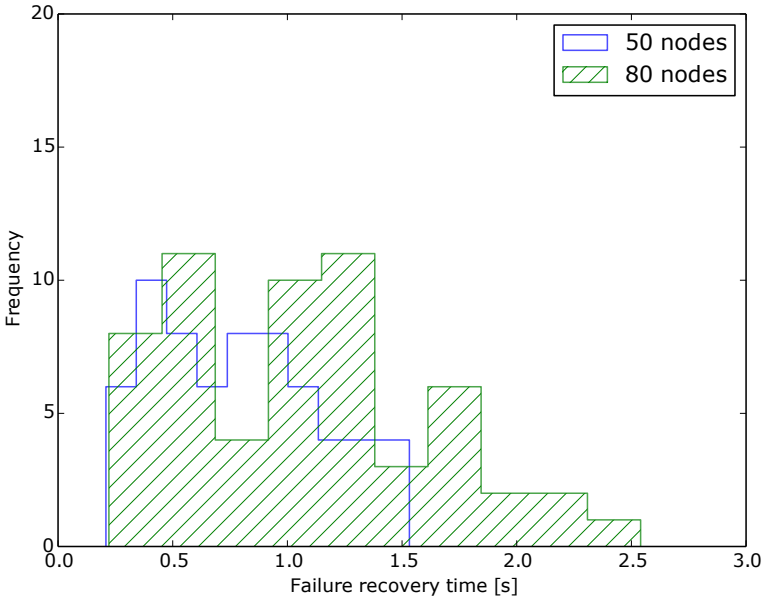


Figure 5.6: Experiment 2: Fail-over time

longer and therefore more rules need to be updated. The measurements show that the concept is viable for all kinds of topologies and fulfilled all the listed requirements.

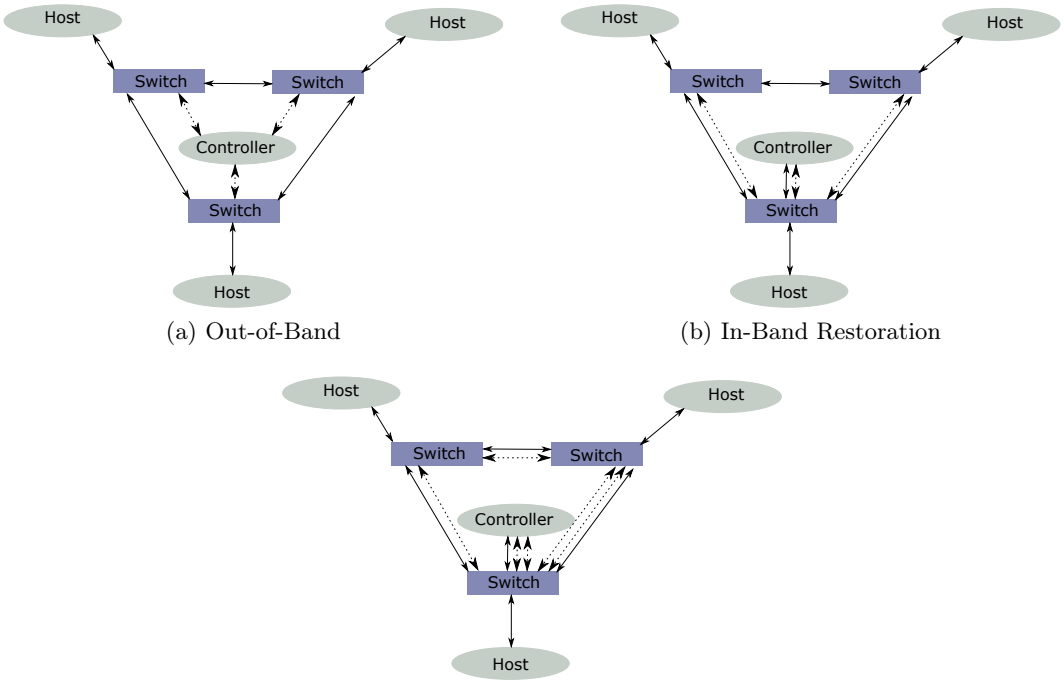
5.2.4 Conclusion

This subchapter presented a self-configuring network solution that can deliver hard quality-of-service guarantees, support a selectable level of redundancy and self-healing features as well as simple plug and play service discovery. A central controller receives service requests and places the requested flows throughout the network. Furthermore, the controller checks registrations beforehand if any existing traffic contract will be violated and rejects the registration in such a case. All accepted flows get installed in the network and a bounded upper latency is guaranteed. Due to duplication on a network level, a link failure in the network does not incur packet loss but only a computation time till the full redundancy level is restored. The solution was shown to work on different randomly generated topologies with similar performance results and is therefore a viable candidate for applications with hard real-time requirements.

5.3 A Network with In-band Configuration

Note This section is part of a publication presented at the 1st International Workshop on Software Defined Networks and Network Function Virtualization (SDN-NFV) at the 4th IEEE International Conference on Software Defined Systems (SDS-2017) in 2017 [125].

In this subchapter we present a concept that allows for deterministic in-band configuration of an OpenFlow network with admission-controlled traffic. Therefore, we introduce a safe channel that enforces traffic characteristics on the in-band control message while getting rid of the out-of-band connection. A model on how to configure the traffic enforcement depending on available network capacity and configuration time constraints is given and evaluated using a proof of concept implementation in Mininet. Further a mathematical model has been introduced that describes upper bounds of configuration times in OpenFlow based on the allocated rate and validated against the proof of concept implementation.



(a) Out-of-Band

(b) In-Band Restoration

(c) In-Band Redundant

Figure 5.7: Different Configuration Concepts

5.3.1 Concept

The concept we present in this subchapter consists of three parts, the bootstrapping process, failure handling and deterministic behavior.

5.3.1.1 Bootstrap Process

As the switches do initially not have a controller connection, a local logic needs to install basic bootstrapping rules into the switch. Such rules have to work, if no controller connection is present. This can be done by installing them with a low OpenFlow priority value, without a timeout, so that they get overwritten as soon as a controller installs a rule with higher priority in the switch's lookup table. We, therefore, use OpenFlow's *normal* operating mode, which allows matched packets to be treated as in a traditional layer 2 switch.

Initialize Switches with OFP_NORMAL

With local L2 switching for rules matching on ARP/DHCP/TCP-SYN only, the first ARP/DHCP request will be sent with a broadcast MAC and query the MAC address of the received or preconfigured OpenFlow controller. While a packet storm in looped topologies might occur, this will only occur on unconfigured parts of the network and it will be stopped as soon as the first switch connects to the controller. Such a packet storm would be avoided completely, if the switch also supports STP or IS-IS protocols.

On Connection Establishment with First Switch

Once a switch is connected rules to overwrite the L2 switching have to be installed. At this point the switch stops the packet storm for its segment. If the previous step was done by using STP, it also needs to be disabled to activate all STP disabled links. Specifically, the controller installs a proxy ARP handler, a rule that sends all ARP to the controller, and push explicit rules onto the switch specifying the controller connection itself. Both rules will be installed with a meter attached that polices traffic according to Section 5.3.1.3.

Connection Establishment for Remaining Switches

All other switches repeat steps 1 and 2. Due to the meshed nature of the network the controller might see multiple messages with TCP-SYN originated from the same switch.

In such a case the controller needs to make sure to only open one control connection per data-plane identifier.

5.3.1.2 Fail-Over

In case of a failure, the concept shall provide safety so that the configuration plane does not affect any operational traffic. We look here at the following failure cases, assuming a single fault per fault containment region.

Babbling Idiot

This failure case is defined as a crashed program that gets stuck in a continuous sending routine thereby flooding the network with invalid or replicated packets. To prevent an impact on the system in case a babbling idiot failure is occurring, the previously mentioned meters are installed to police traffic according to its predefined traffic characteristics (see Chapter 5.3.1.3) and thereby constraining any babbling idiot fault to its own fault containment region.

Switch or Link Failure

Neighboring switches will generate a controller event, so that the controller can act on this information and run a restoration algorithm on the network to restore control connections, e.g. see Figure 5.7. Switches directly affected by a link failure will drop all explicit rules on the disabled ports and revert back to L2 switch rules as presented in step 1 of Section 5.3.1.1, until they are reconnected to the controller.

Another option to be even faster during recovery is a redundant communication channel. While in [100] fast failover buckets have been proposed, such a configuration only works for small fully meshed networks as link failures have to occur on a switch with a second reserved link available. In larger more realistic networks, we propose here preventive duplication of control traffic as seen in Figure 5.7(c). Forwarding will take place on two independent paths by using OpenFlow's duplication mechanisms at the edges. As OpenFlow, however, does not offer a deduplication mechanism we rely on TCP to filter out duplicated and out of order packets. We have implemented this in our proof-of-concept implementation and tested it successfully for link failures with zero fail-over time for the control channel. For practical use, however, deduplication should be supported on the network layer like in IEEE TSN to avoid deduplication issues like out of order delivery of the packets.

5.3.1.3 Determinism of Safe Control Channel

We define a control channel to be deterministic, if the amount of traffic it transmits through the network is limited in all cases and the configuration time is upper bounded. We achieve such deterministic behavior in our control channel by describing all traffic beforehand and then enforcing compliance to such descriptions. In Chapter 4.2 we presented how this is done for non-control traffic, by allowing only registered traffic into the network and using formal verification in the form of network calculus [40] to guarantee latency and buffer requirements. Therefore, also control traffic needs to be subject to characterization and traffic limitation. While enforcing periodic UDP connections is quite straight forward with network calculus, OpenFlow relies on TCP as its transport protocol. TCP connections, however, are more complicated to model due to feedback loops, e.g. window-based flow control and congestion control as implemented by TCP.

While there is some work on this like [57, 126], we take advantage of the fact, that control traffic is not considered critical traffic. We thus only need to protect the network from exhaustive use of the control connection and influence on other network services. Packet loss may occur as retransmissions will take place anyway in TCP.

We, therefore, use over-provisioning of the control channel and enforce traffic per switch connection with a token bucket style mechanism and network calculus for verification. The burst size of our token bucket will be set to slightly above the average size of an OpenFlow control message, so that fast reaction times can still occur at the expense of slowed down complex operations. With a careful controller design avoiding such large operations the control channel, therefore, also offers bounded latency for all control messages and thus bounded configuration times. An experimental evaluation on the rate parameter for such policed control channels is done in Section 5.3.4.

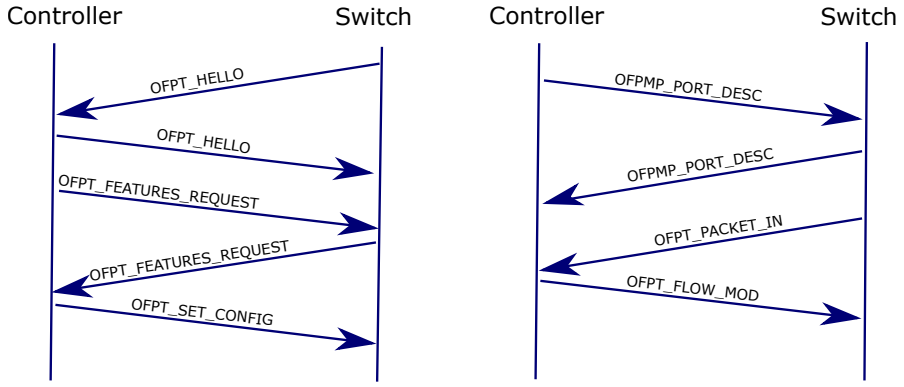


Figure 5.8: OpenFlow 1.3 Message Transactions

Figure 5.8 gives an overview of OpenFlow messages that are exchanged during bootstrap and recovery phases of a typical OpenFlow session. As mentioned, the messages are

dynamic in size, where typical control messages are below 80 Bytes payload for hello messages and feature requests. A flow-mod operation or even a packet-out message, however, can be of arbitrary size. Bigger messages as such have to be split into multiple smaller ones by either the controller software or the operating system. An overview of communication usage of the control channel can be seen in Figures 5.9, 5.10 and 5.11.

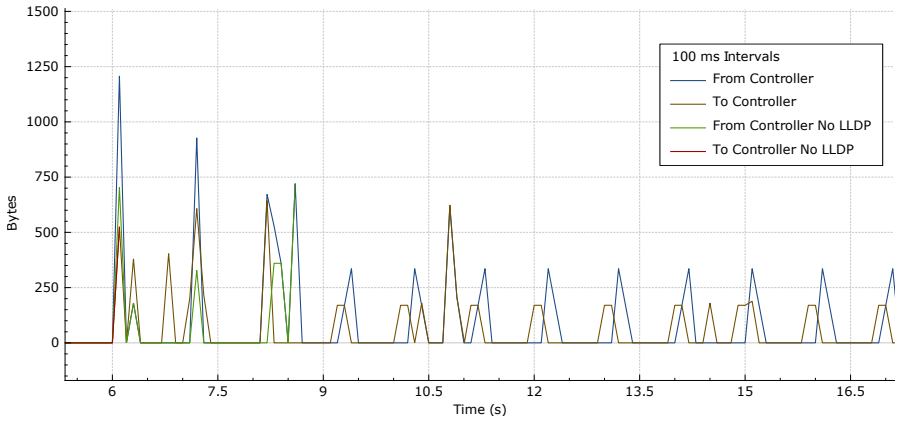


Figure 5.9: OpenFlow 1.3 Connection Bootstrap (100ms resolution)

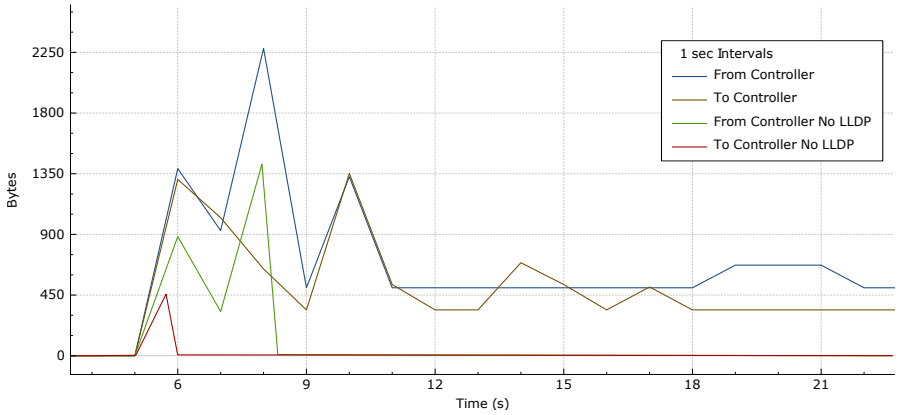


Figure 5.10: OpenFlow 1.3 Connection Bootstrap (1s resolution)

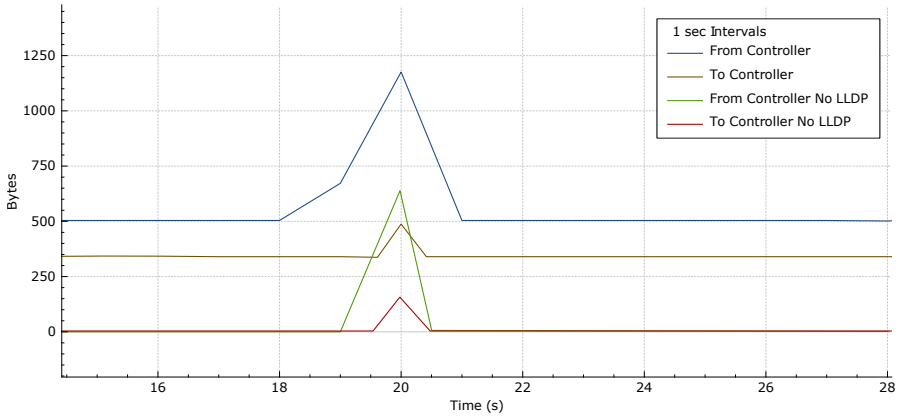


Figure 5.11: Traffic behaviour during fault and reconfiguration (1s resolution)

5.3.2 Model

In this section we define a model that gives the bootstrap time depending on the assigned rate for each control channel. While we use over-provisioning here, the evaluation has shown, that the assigned rate should be above 50 Bytes/sec for reasonable reaction times and about 200 Bytes/sec for fast reaction times and no congestion effects.

For the model below, we define the constant t_{boot} as the time the switch needs to react once the link is up. Further we define $d(r)$ as the network calculus (see Chapter 2.4.1) delay function calculating the queuing delays in the network. As the arrival curve for NC we use $\alpha(t) = r * t + b$ with r being the rate and b the burst size of 200 Bytes.

Next we define the partial bootstrap time $x(r)$ in Equation (5.1) for a single switch without congestion/shaping or limitations. We add three times the queuing delay from the controller to the server and twice the delay back to the general transmission time of the shaped output queue. While this does not consider the bidirectional characteristics of a communication channel, it gives us a good estimate for the bootstrap time.

$$\begin{aligned} x(r) = t_{\text{boot}} &+ \frac{\sum \text{OpenFlow messages}}{r} \\ &+ 3 \cdot d_{\text{controller}\rightarrow\text{switch}}(r) \\ &+ 2 \cdot d_{\text{switch}\rightarrow\text{controller}}(r) \end{aligned} \tag{5.1}$$

Based on this we can calculate the delay caused by shaping the outgoing queue T_0 for one switch, depending on the rate r, see Equation (5.2).

$$T_0(r) = \begin{cases} x(r), & r \geq 250 \\ x(r) + \frac{(\sum \text{Opnf messages}) - r}{0.2e{-}3}, & 50 \leq r \leq 250 \\ \infty, & r \leq 50 \end{cases} \tag{5.2}$$

Finally, we can sum up the bootstrap time for the individual switches based on a depth N of the switch, see Equation (5.3):

$$T_{N_{\text{switches}}}(r) = \sum_{N_{\text{switches}}} T_0(r) \tag{5.3}$$

5.3.3 Implementation

The proposed concept has been implemented as a proof of concept in Mininet [124] with separated network namespaces in order to separate traffic. Open vSwitch [127] was used

as OpenFlow switch due to its easy integration into Linux. However, the needed meters for traffic policing are not supported in Open vSwitch.

While Open vSwitch does not enforce issued meters [128], the implementation, however, does accept and reply to meter commands so that a transparent implementation for future use on real switches was possible.

As OpenFlow controller, Ryu [123] was used. Ryu is an open source Python based controller which allows for easy extensions through its modular design. To make sure our control traffic is limited as needed for the evaluation in Section 5.3.4, we applied two mechanisms to make up for Open vSwitch's inability to limit traffic. First we apply traffic shaping in the controller, by limiting Ryu's outgoing queue transmission rate, limited by a token bucket implementation. Further, for traffic policing along the way, we used a workaround of Linux's Traffic Shaping (TC) tool which allows for easy traffic policing.

5.3.3.1 Configure Policing

The rate limiter's burst size within the switches and Ryu's outgoing queue's burst size were set to match the largest regular OpenFlow message, e.g. a simple flow modification of 200 Bytes. By doing so, all simple control messages will pass and more complex messages will be fragmented or delayed. Fragmentation is automatically done by TCP, as we set the TCP maximum segment size (MSS) to its respective burst size.

5.3.3.2 Topology Discovery & Route Placement

Once the first switch is installed, the switch starts broadcasting OFDP messages out of its ports in order to detect neighboring network devices. The resulting packet-in events at the controller will allow Ryu to build up a topology view of the network. This information is then used to allow shortest path calculation for any further switches that register at the controller.

5.3.4 Evaluation

In this section we evaluate our proposed solution in terms of bootstrapping times, fail-over times and the rate limitations in order to be able to judge how much influence the safe channel has on the overall configuration. All experiments have been run 50 times with different random seeds.

5.3.4.1 Bootstrap Times

The first evaluation looks at bootstrap times to get a better understanding on how the safe channel behaves and reacts. We therefore evaluate three different topologies in Mininet with different numbers of switches. We use typical industrial devices, where a node will have a switch included in the housing to allow for daisy-chained line replaceable units without any extra parts. Therefore, each switch has one node connected. The controller here is directly connected to the first switch. The mesh topology is a fully-meshed topology. Figure 5.12 gives an overview of the measured boot times.

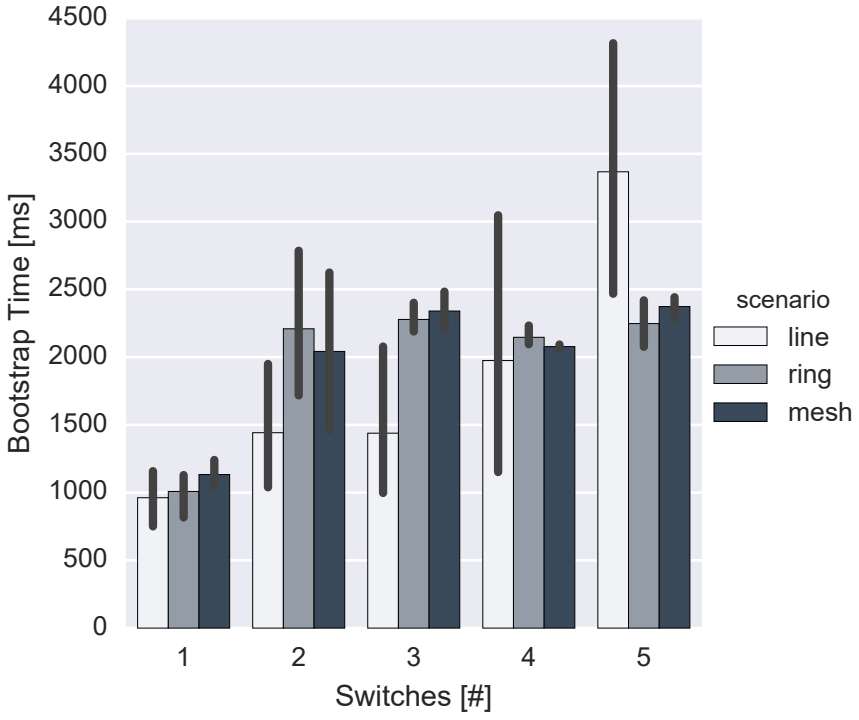


Figure 5.12: Time till bootstrapping depending on configuration
(40 runs per scenario were performed)

It can be seen in Figure 5.12 that the initial bootstrap of the first switch takes around one second. This number is quite high and caused by the use of our prototype software, which waits a certain time after link-up. We expect this value to be much lower on real hardware devices. Adding a second switch to the network, all values increase, however, by a smaller value for the line topology and a larger one for mesh and ring. This can be explained by the loop in the topology, which needs to be overcome first and similarly multiple registration requests at the controller due to duplication at the network layer. For three, four and five switches in the network, the resulting bootstrap times are similar. However, the line topology shows a higher deviation from the average. A last switch in the line can only be configured when all other switches are well initialized. The mesh and ring scenario offer here alternatives paths that can be used to speed up this process.

5.3.4.2 Fail-Over

In the next evaluation we look at failures in the network, specifically at link failures. Figure 5.13 shows the times from occurrence of a link failure, notification of the controller, calculation of backup paths to re-installation of a new control connection. The simulation was run on a fully meshed network of 5 switches, each with one host connected, and randomly broken links that carried control channels before the fault. Redundant and out-of-band scenarios were not considered due to their infinite and zero re-configuration times.

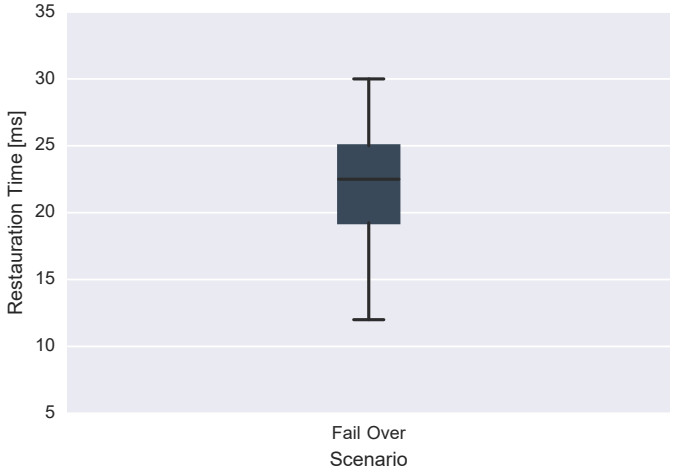


Figure 5.13: Restoration scenario for fully meshed network
(40 runs were performed)

It can be seen in Figure 5.13, that reconfiguration occurs quite fast within the network with a mean value of around 23ms. Most differences come from the fact whether one or more configuration channels had to be rerouted. A scenario with a link failure and no impact on control traffic was not considered.

5.3.4.3 Determinism

The last part of this evaluation looks at the determinism of our safe channel. We therefore limited the rate that was allowed to pass through the safe channel and measured again the bootstrap time. It can be seen in Figure 5.14 that a rate of about 100 Bytes/sec per switch gives already stable, usable bootstrap times. Congestion disappears at about 200 Bytes/sec per switch which also gives similar measurement times as in the unlimited scenario in Figure 5.12. A regular OpenFlow message is about 100 Bytes in size. Once adding the LLDP discovery feature, a further increase in rate usage can be seen, which

takes up to 100 Bytes/sec per port. This is correlating to one LLDP out message per switch port per second.

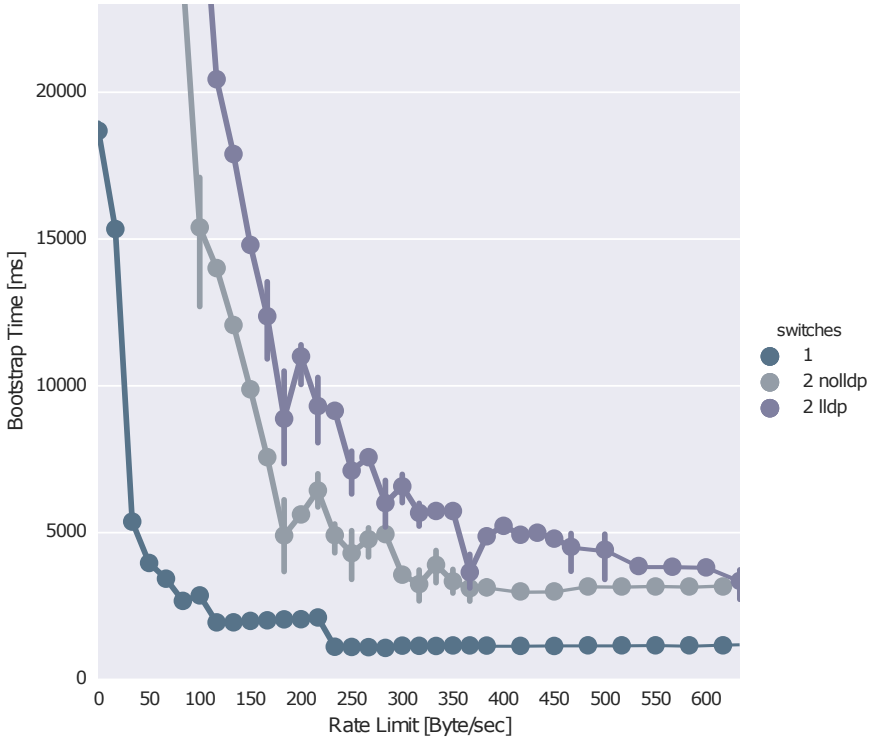


Figure 5.14: Influence of rate limit on bootstrap time

To further understand this simulation we further measured bandwidth utilization per safe channel as seen in Figures 5.9, 5.10 and 5.11. These measurements also reveal that a rate of 100 Bytes/sec will be enough for all bootstrap or reconfiguration cases with some slow down, and no slow down at all above 200 Bytes/sec.

The case for restoration is more complex. During restoration large messages are generated by the controller, containing new route information for the switches. Such messages are often larger than 200 Bytes and will, therefore, be fragmented, thus, slowing down reconfiguration. While this slowdown is hard to quantify in random generated topologies, such a value has to be carefully picked in the designed network that is to be redundant. As a rough value, each new route information will cost about 30 Bytes per switch. If multiple routes are to be rerouted on a single switch, the throttle value can be adjusted. It should be noted, that reconfiguration will, nevertheless, work in the throttled case too with degraded convergence time.

5.3.5 Conclusion

In this subchapter we presented a safe in-band configuration concept for deterministic OpenFlow configuration in industrial Ethernet networks. The channel allows to avoid external out-of-band configuration networks while at the same time offering a guarantee that a faulty controller or switch will not have an impact on the operational functionality of the network. A concept was presented, implemented as proof of concept and evaluated. Further a mathematical model was developed to describe rate requirements of the configuration channel for a given upper configuration time. The proof of concept was used during evaluation. It was shown, that a rate of about 100 Bytes/sec per control connection is sufficient to bootstrap an OpenFlow network and reconfigure it in case of faults.

5.4 Results & Evaluation

In this chapter we presented two concepts for a self-configuring real-time network based on the aeronautical requirements from Chapter 3.1. Each concept aimed to fill gaps found in current state of the art literature, to push the area of self-configuration mechanisms forward. An overview of each subchapter's contribution is summarized in Table 5.2.

	Configuration Process						
Self-Configuration Mechanism	In-Band Management	Online Service Discovery	Reconfigurable at run-time	Isolation from Data Traffic	COTS Support	QoS on Operational Traffic	QoS on Operational COTS Traffic
Dürkop et al. [107][108]	n	y	**n**	n[1]	**n**	**y**	**n**
Heise et al. [120]	n	y	**y**	n[2]	**y**	**y**	**y**
Sharma et al. [100]	y	**n/a**	y	**n**	y	**n**	**n**
Heise et al. [125]	y	**y**	y	**y**	y	**y**	**y**

Table 5.2: Contribution of this Chapter

5.4.1 Evaluation

In the first part of this chapter we presented a solution to achieve a self-configuring real-time network. The solution can deliver hard quality of service guarantees by implementing our mechanisms from Chapter 4.2, while also supporting a selectable level of redundancy and self-healing features as well as simple plug and play service discovery. It is based on a central controller that receives service requests and places the requested flows throughout the network. The solution was shown to work on all kinds of different randomly generated topologies with similar performance results and is therefore a viable candidate for applications with hard real-time requirements. On the downside, the solution was based on OpenFlow which uses an out-of-band configuration channel that infers extra cabling and installation costs.

[1] Configuration before Operation
[2] Through Out-of-Band Network

In Chapter 5.3 we aimed to remove this shortcoming by introducing a safe in-band configuration channel for OpenFlow switches. The channel is considered safe, as all configuration traffic is subject to traffic policing and will thus not interfere with any operational traffic in the network. A concept was presented, implemented as proof of concept and evaluated. Further a mathematical model was developed to describe rate requirements of the configuration channel for a given upper configuration time. The proof of concept was used during evaluation and it was shown that the models hold the upper bounds for configurations. For this to be used in certification, however, an analytic bound for fault recovery has to be presented and this chapter's work extended accordingly.

5.4.2 Interpretation

In this section we interpret the results achieved in this chapter and put them into context of a future avionic network.

Chapter 5.2 presented a plug and play discovery service. Such a service could offer great benefits in production lines of airplanes and for changes of aircraft configurations. The idea can be easily extended for multiple devices and adopted to any kind of network that can offer guarantees. Through its central design, all unknown traffic has to register and thus will also support devices that are not manufacturer controlled. A problem, however, arises with the configuration process of aircraft as presented in Chapter 5.1. As some systems are subject to certification, so is the network configuration. To solve this problem, the algorithm admitting and allocating resources to the network, e.g. the core of the plug and play system will be subject to certification and rigid development rules and guidelines. While this could be feasible, this was not in the scope of this thesis.

In Chapter 5.3 we extended this work with a safe in-band configuration channel. In-band configuration is a prerequisite to actually be applicable to a real aircraft use-case. As the in-band channel was designed in a safe way that does not influence operational traffic even during faults, the proposed concept is likely to be actually considered in next generation aircraft networks.

Chapter 6

Conclusion & Future Work

This chapter summarizes and concludes with the contents presented in this thesis. We put the content in context to the research questions presented in Chapter 1 followed by an overview of future research directions.

6.1 Summary & Conclusion

This thesis was centered around two research problems, which we focused on while developing our solutions and interpreting its results.

Research problem 1: Dependable networks that satisfy hard quality of service requirements and support seamless changes in its configuration at run-time.

In Chapter 3.2 existing real-time networks were presented and compared to each other. They were then evaluated against aeronautical requirements. It was shown, that none of the current networks could fulfill all requirements needed and a research gap did exist in terms of reliability and ease of configuration. Chapter 4, therefore, proposed three different mechanisms and concepts to fill these gaps by introducing a deterministic ring in Chapter 4.1 with mixed criticality classes and a deterministic software-defined networking solution in Chapter 4.2. The latter solution was evaluated to be technologically capable to handle a current state of the art avionic switch configuration while offering ease of configuration. Finally, we presented an open-sourced IEEE TSN simulation framework to evaluate the currently stable features of TSN and proposed a model profile that allows the production of avionic chips only covering a small subset of TSN in order to be easier certifiable.

Research problem 2: Means for assured self-configuration of networked devices with guaranteed properties and timing of the ensuing systems.

In Chapter 3.3 we demonstrated that no current self-configuration solution can configure all parts of a network according to our requirements from Chapter 3.1. Most solutions only configured loop-free topologies or proprietary network systems. We, therefore, introduced in Chapter 5.2 a protocol that allows for plug and play service of a multitude of devices on standard commercially available OpenFlow switches. The problem with this solution, however, was the need for an out-of-band network to configure OpenFlow switches. This, however, would be impractical in long aircraft cabins. The concept was therefore extended in Chapter 5.3 to host a safe in-band configuration channel that gets rid of the out-of-band network and does not impact operational traffic. Both solutions combined offer a good starting point for the development of a real-time self-configuring future avionic network that we hope to see flying soon.

6.2 Future Research Directions

In this thesis we studied the effects of delays in networks and found means to circumvent such. However, we did not consider effects that happen inside the software nor inside processing logic within switches and computational systems. Such are ever more tightly integrated, so called System-on-a-Chip (SoCs). These SoCs could be connected to a real-time network and would need to be considered in an end-to-end evaluation.

While our solution is based on OpenFlow, the idea of software-defined networking is currently being extended with programmable elements at ingress and egress of the switch to allow rapid implementation of different scheduling techniques. At the forefront of this is the P4 language [129] that would offer even more possibilities on a switch in terms of management and traffic control algorithms. This could also be used to implement different and new self-configuration mechanisms that rely less on a central controller and could still offer the dependability needed.

One problem mentioned in Chapter 2.2 is the need for certification in avionics. While our solution offers easy configuration and eases all work surrounding it, the final network will still need to be certified if it is to be used in critical networks. While one way would be to turn off self-configuration after an initial configuration and offline-prove that the configuration is working, an alternative solution is the certification of the configuration source code itself. This is in theory possible, but it will also mean a lot of restriction on dynamic allocations and also limit of future proof features as the configuration code will not be update-able in fast intervals.

Appendix

Appendix A

Author's Contribution

Heise, P., Geyer, F., Elefsiniotis, A., and Obermaisser, R. (2015a). Towards a Deterministic Mixed Criticality High Availability Seamless Redundancy (HSR) Ring. In *8th International Workshop on Communication Technologies for Vehicles (Nets4Cars/Nets4Trains/Nets4Aircraft)*.

Heise, P., Geyer, F., and Obermaisser, R. (2015b). Deterministic OpenFlow: Performance Evaluation of SDN Hardware for Avionic Networks. In *2nd International Workshop on Management of SDN and NFV Systems at 11th International Conference on Network and Service Management (CNSM)*.

Heise, P., Geyer, F., and Obermaisser, R. (2016a). TSimNet: An industrial Time Sensitive Networking simulation framework based on OMNeT++. In *8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*.

Heise, P., Geyer, F., and Obermaisser, R. (2017). Self-Configuring Deterministic Network with In-Band Configuration Channel. In *1st International Workshop on Software Defined Networks and Network Function Virtualization (SDN-NFV) at Fourth IEEE International Conference on Software Defined Systems (SDS-2017)*.

Heise, P., Lasch, M., Geyer, F., and Obermaisser, R. (2016b). Self-Configuring Real-Time Communication Network based on OpenFlow. In *22nd IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*.

Heise, P., Tobeck, N., Hanka, O., and Schneele, S. (2014). SAFDX : Deterministic High-Availability Ring for Industrial Low-Cost Networks. In *7th International Workshop on Communication Technologies for Vehicles (Nets4Cars-Fall)*.

Steiner, W., Heise, P., and Schneele, S. (2014). Recent IEEE 802 Developments and Their Relevance for the Avionics Industry. In *33rd Digital Avionics Systems Conference (DASC)*.

Wichtlhuber, M., Heise, P., Scheurich, B., and Hausheer, D. (2013). Reciprocity with virtual nodes: Supporting mobile peers in Peer-to-Peer content distribution. In *9th International Conference on Network and Service Management (CNSM)*.

Wichtlhuber, M., Heise, P., Scheurich, B., Rückert, J., and Hausheer, D. (2014). vINCENT : An Incentive Scheme Supporting Heterogeneity in Peer-to-Peer Content Distribution. In *39th Conference on Local Computer Networks (LCN) (Best Paper Award)*.

Appendix B

Bibliography

[1] IEEE 802.1 Working Group. Time Sensitive Networks. URL http://www.ieee802.org/1/pages/tsn.html.

[2] Jean-Claude Laprie and Colonel Roche. Dependable Computing: Concepts, Limits, Challenges. In *Special Issue of the 25th International Symposium On Fault-Tolerant Computing*, 1995.

[3] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. *Fundamental Concepts of Dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[4] Jean-Bernard Itier. A380 Integrated Modular Avionics, 2007. URL http://www.artist-embedded.org/docs/Events/2007/IMA/Slides/ARTIST2_IMA_Itier.pdf.

[5] Ian Moir, Allan Seabridge, and Malcolm Jukes. *Civil Avionics Systems*. John Wiley Sons, 2013.

[6] Cary R Spitzer, Uma Ferrel, and Thomas Ferrel. *Digital Avionics Handbook*. CRC Press, 2014.

[7] Juan R Pimentel. Safety-Reliability of Distributed Embedded System Fault Tolerant Units. In *29th Annual Industrial Electronics Society (IECON)*, 2003.

[8] Elena Dubrova. *Fundamentals of dependability*. Springer, 2013.

[9] Andrew Kornecki and Janusz Zalewski. Software Certification for Safety-Critical Systems: A Status Report. In *International Multiconference on Computer Science and Information Technology*, 2008.

[10] SAE International. Aerospace Recommended Practice ARP-4754A (Guidelines For Development Of Civil Aircraft and Systems), 2010.

[11] Radio Technical Commission for Aeronautics (RTCA). *Software considerations in airborne systems and equipment certification (DO 178).* RTCA Inc, Washington, 1992.

[12] Radio Technical Commission for Aeronautics (RTCA). *Design Assurance Guidance for Airborne Electronic Hardware (DO 254).* RTCA Inc, Washington, 2000.

[13] Federal Aviation Administration. *FAA System Safety Handbook.* 2000.

[14] SAE International. Aerospace recommended practice ARP-4761 (Guidelines and Methods on conducting the safety assessment process on civil airborne systems and equipment). 2008.

[15] Nikhil Balakrishnan. An Overview of System Safety Assessment. In *Dependability in Medicine and Neurology: Using Engineering and Management Principles for Better Patient Care.* Springer International Publishing, Cham, 2015.

[16] Scott L Pelton and K D Scarbrough. Boeing systems engineering experiences from the 777 AIMS program. *IEEE transactions on aerospace and electronic systems*, 1997.

[17] Radio Technical Commission for Aeronautics (RTCA). *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations (DO 297).* Washington, 2005.

[18] Aeronautical Radio Incorporated. Avionics Application Software Standard Interface (ARINC 653). 2008.

[19] Alan Burns and Robert I Davis. Mixed Criticality Systems - A Review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[20] Sanjoy Baruah, Haohan Li, and Leen Stougie. Towards the design of certifiable mixed-criticality systems. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010.

[21] Michael Paulitsch, Oscar Medina Duarte, Hassen Karray, Kevin Müller, Daniel Münch, and Jan Nowotsch. Mixed-Criticality Embedded Systems–A Balance Ensuring Partitioning and Performance. In *Euromicro Conference on Digital System Design (DSD)*, 2015.

[22] Guoqing Wang and Qingfan Gu. Research on distributed integrated modular avionics system architecture design and implementation. In *32nd Digital Avionics Systems Conference (DASC)*, 2013.

[23] Björn Annighöfer and Frank Thielecke. *Supporting the Design of Distributed Integrated Modular Avionics Systems with Binary Programming.* Deutsche Gesellschaft für Luft-und Raumfahrt-Lilienthal-Oberth eV, 2013.

[24] European Aviation Safety Agency (EASA) Software Complex Electronic Hardware section. Certification Memorandum (SWCEH-001): Development Assurance of Airborne Electronic Hardware. 2011.

[25] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications.* Springer Science & Business Media, 2011.

[26] Giorgio C. Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications.* Springer Science Business Media, 2011.

[27] Paolo Pedreiras and Luis Almeida. Approaches to Enforce Real-Time Behavior in Ethernet. In *Industrial communication technology handbook.* CRC Press, 2005.

[28] Stan Schneider. Making Ethernet work in real time. *Sensors Magazine*, 17(11), 2000.

[29] Gerard Le Lann and Nicolas Rivierre. Real-Time Communications over Broadcast Networks: the CSMA-DCR and the DOD-CSMA-CD Protocols (RR-1863). Technical report, INRIA, 1993.

[30] Junghoon Lee. A Variable Bandwidth Allocation Scheme for Ethernet-Based Real-Time Communication. In *International Conference on Real-Time Computing Systems and Applications*, 1994.

[31] Paulo Pedreiras, Luis Almeida, and Paolo Gai. The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency. In *14th Euromicro Conference on Real-Time Systems*, 2002.

[32] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, 2008.

[33] Ricky W Butler. Fault-tolerant clock synchronization techniques for avionics systems. In *AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference*, Atlanta, Georgia, 1988.

[34] Chitra Venkatramani. *The design, implementation and evaluation of RETHER: a real-time Ethernet protocol.* PhD thesis, State University of New York at Stony Brook, 1996.

[35] Aeronautical Radio Incorporated. Aircraft Data Network, Part 5: Network Domain Characteristics and Interconnection. 2005.

[36] Fabien Geyer. *End-to-End Flow-Level Quality-of-Service Guarantees for Switched Networks*. PhD thesis, Technische Universität München, 2015.

[37] Anne Bouillard, Laurent Jouhet, and Éric Thierry. Tight performance bounds in the worst-case analysis of feed-forward networks. In *Proceedings - IEEE INFOCOM*, 2010. ISBN 9781424458363. doi: 10.1109/INFCOM.2010.5461912.

[38] Louise E Moser and P M Melliar-Smith. Probabilistic bounds on message delivery for the Totem single-ring protocol. In *Real-Time Systems Symposium*, pages 238–248. IEEE, 1994.

[39] Rene L Cruz. A calculus for network delay. *IEEE Transactions on information theory*, 1991.

[40] Jean-Yves Le Boudec and Patrick Thiran. Network calculus: a theory of deterministic queuing systems for the internet. In *Vol. 2050. Springer Science Business Media*. 2001.

[41] Jean-Yves Le Boudec and Patrick Thiran. A short tutorial on network calculus. I. Fundamental bounds in communication networks. In *International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century*, 2000.

[42] Jean-Yves Le Boudec and Patrick Thiran. A short tutorial on network calculus. II. Min-plus system theory applied to communication networks. In *Circuits and Systems (ISCAS)*, 2000.

[43] Fabrice Frances, Christian Fraboul, and Jérôme Grieu. Using network calculus to optimize the AFDX network. In *3rd European Congress ERTS Embedded real-time software*, 2006.

[44] Steffen Bondorf and Jens B Schmitt. The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus. In *8th International Conference on Performance Evaluation Methodologies and Tools*, 2014.

[45] Rajeev Alur and David L Dill. Study A theory of timed automata. *Theoretical computer science*, 1994.

[46] Hussein Charara, Jean-Luc Scharbarg, Jerome Ermont, and Christian Fraboul. Methods for bounding end-to-end delays on an AFDX network. *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 193–202, 2006.

[47] Steven Martin and Pascale Minet. Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, 2006.

[48] Henri Bauer, Jean Luc Scharbarg, and Christian Fraboul. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. *IEEE Conference on Emerging Technologies and Factory Automation (ETFA 09)*, 2009.

[49] A. Taber and E. Normand. Single Event Upset in Avionics. *IEEE Transactions on Nuclear Science*, 1993.

[50] International Electrotechnical Commission. IEC 61508 - Funktionale Sicherheit sicherheitsbezogener elektrische/elektronischer/programmierbarer Systeme. Technical report, 2002.

[51] Mohammed Abuteir and Roman Obermaisser. Mixed-Criticality Systems Based on Time-Triggered Ethernet with Multiple Ring Topologies. *9th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2014.

[52] Jorge Munoz-Castaner, Rafael Asorey-Cacheda, Felipe J Gil-Castineira, Francisco J Gonzalez-Castano, and Pedro S Rodriguez-Hernandez. A Review of Aeronautical Electronics and its Parallelism with Automotive Electronics. *IEEE Transactions on Industrial Electronics*, 2011.

[53] Aeronautical Radio Incorporated. Aircraft Data Network, Part 7: Avionics Full Duplex Switched Ethernet (AFDX). 2006.

[54] Jens B. Schmitt, Frank A. Zdarsky, and Ivan Martinovic. Improving performance bounds in feed-forward networks by paying multiplexing only once. In *Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB)*, 2008.

[55] Marc Boyer and David Doose. Combining network calculus and scheduling theory to improve delay bounds. In *20th International Conference on Real-Time and Network Systems (RTNS 12)*, New York, New York, USA, 2012. ACM Press.

[56] Frederic Ridouard, JL Scharbarg, and Christian Fraboul. Stochastic upper bounds for heterogeneous flows using a Static Priority Queueing on an AFDX network. *Journées FAC*, 2008.

[57] Fabien Geyer, Stefan Schneele, and Georg Carle. Towards stochastic flow-level network modeling: Performance evaluation of short TCP flows. In *Local Computer Networks (LCN)*, 2014.

[58] Emanuel Heidinger and Nils Kammenhuber. Network Calculus and mixed-integer LP applied to a switched aircraft cabin network. In *Quality of Service (IWQoS)*, 2012.

[59] Ahmad Al Sheikh, Olivier Brun, Maxime Ch, and Pierre-Emmanuel Hladik. Optimal Design of Virtual Links in AFDX Networks. *Real-Time Systems*, 2013.

[60] Youngjun Cha, Juho Lim, Sunyoung Lee, Dongjin Kim, and Ki Il Kim. New feasible grouping algorithms for virtual link in AFDX networks. In *4th International Conference on Informatics, Electronics and Vision (ICIEV 2015)*, 2015.

[61] Hermann Kopetz and Günter Grünsteidl. TTP - A time-triggered protocol for fault-tolerant real-time systems. *23rd International Symposium on Fault-Tolerant Computing (FTCS 23)*, pages 524–533, 1993.

[62] Richard Zurawski. *Networked embedded systems*. CRC Press, 2009.

[63] Hermann Kopetz. A comparison of CAN and TIP. *Annual Reviews in Control*, 2000.

[64] Christopher Temple. Avoiding the babbling-idiot failure in a time-triggered communication system. *Fault-Tolerant Computing*, 1998.

[65] SAE International. SAE AS6802 Time-Triggered Ethernet, 2011.

[66] Michael Lauer, John Mullins, and Moez Yeddes. Cost optimization strategy for iterative integration of multi-critical functions in IMA and TTEthernet architecture. *Proceedings - International Computer Software and Applications Conference*, 2013.

[67] Kamarajugadda Padmanabham, Prabhakar Kanugo, K Nagabhushan Raju, and M Chandrashekar. A Review of MIL-STD-1553 Bus Trends and Future. *International Journal of Advanced Research in Computer and Communication Engineering*, 2016.

[68] Yasemin Isik. ARINC 629 data bus standard on aircrafts. In *9th WSEAS international conference on Circuits, systems, electronics, control signal processing*, 2010.

[69] Neil C. Audsley and Alan Grigg. Timing analysis of the ARINC 629 databus for real-time applications. *Microprocessors and Microsystems*, pages 1–11, 1997.

[70] International Electrotechnical Commission. IEC 62439-3 High Availability Automation Networks - PRP HSR, 2012.

[71] Saad Allawi Nsaif and Jong Myung Rhee. Improvement of High-availability Seamless Redundancy (HSR) Traffic Performance. In *14th International Conference on Advanced Communication Technology (ICACT)*, 2012.

[72] Saad Allawi Nsaif and Jong Myung Rhee. RURT: A novel approach for removing the unnecessary redundant traffic in any HSR closed-loop network type. *International Conference on ICT Convergence (ICTC)*, 2013.

[73] Kyusang Lee, Yonggyu Lee, Younghyun Kim, and June-koo Kevin Rhee. Efficient Network Design for Highly Available Smart Grid Communications. In *Opto-Electronics and Communications Conference (OECC)*, 2012.

[74] Nguyen Xuan Tien and Jong Myung Rhee. A Novel Approach for Reducing HSR Unicast Traffic Using Pre-defined Dual Paths. In *10th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, 2016.

[75] Christian Boiger. Class A Bridge Latency Calculations. *IEEE 802 November Plenary Meeting*, (November), 2010. URL http://www.ieee802.org/1/files/public/docs2010/ba-boiger-bridge-latency-calculations.pdf.

[76] Jahanzaib Imtiaz, Jürgen Jasperneite, and Lixue Han. A performance study of Ethernet Audio Video Bridging (AVB) for Industrial real-time communication. In *IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, 2009.

[77] Emanuel Heidinger, Fabien Geyer, Stefan Schneele, and Michael Paulitsch. A performance study of Audio Video Bridging in aeronautic Ethernet networks. In *7th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2012.

[78] Hyung-Taek Lim, Daniel Herrscher, and Telemotive Ag. Performance Comparison of IEEE 802.1Q and IEEE 802.1 AVB in an Ethernet-based In-Vehicle Network. In *8th International Conference on Computing Technology and Information Management (ICCM)*, 2012.

[79] Giuliana Alderisi, Gaetano Patti, and Lucia Lo Bello. Introducing support for scheduled traffic over IEEE audio video bridging networks. In *18th Conference on Emerging Technologies Factory Automation (ETFA)*, 2013.

[80] Open Networking Foundation. OpenFlow Switch Specification, 2012. URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.3.pdf.

[81] Balázs Sonkoly and András Gulyás. On QoS Support to Ofelia and OpenFlow. In *Workshop on Software Defined Networking (EWSDN)*, 2012.

[82] Mohamed Said Seddiki, Muhammad Shahbaz, Sean Donovan, Sarthak Grover, Miseon Park, Nick Feamster, and Ye-Qiong Song. FlowQoS: Providing Per-Flow Quality of Service for Broadband Access Networks. In *Hot topics in Software Defined Networking (HotSDN 14)*, 2014.

[83] Hilmi E. Egilmez, S. Tahsin Dane, K. Tolga Bagci, and A. Murat Tekalp. OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks. In *Signal Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2012.

[84] Keqiang He, Junaid Khalid, Sourav Das, Aaron Gember-Jacobson, and Chaithan Prakash. Latency in Software Defined Networks: Measurements and Mitigation Techniques. In *International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS)*, 2015.

[85] Siamak Azodolmolky, Reza Nejabati, Maryam Pazouki, and Philipp Wieder. An Analytical Model for Software Defined Networking : A Network Calculus-based Approach. In *Global Communications Conference (GLOBECOM)*, 2013.

[86] Adam L. Jacobs, J. Wernicke, Sarp Oral, B. Gordon, and Alan D George. Experimental characterization of QoS in commercial ethernet switches for statistically bounded latency in aircraft networks. In *Local Computer Networks (LCN)*, 2004.

[87] Wang Yun-sheng. The COTS based IMA prototype for hosted applications development. *Digital Avionics Systems Conference (DASC)*, 2012.

[88] Stefan Schneele and Fabien Geyer. Comparison of IEEE AVB and AFDX. In *31st Digital Avionics Systems Conference (DASC)*, 2012.

[89] Till Steinbach, Hyung-Taek Lim, Franz Korf, Thomas C Schmidt, Daniel Herrscher, and Adam Wolisz. Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In *Vehicular Technology Conference (VTC Fall)*, 2012.

[90] Wilfried Steiner, Peter Heise, and Stefan Schneele. Recent IEEE 802 Developments and Their Relevance for the Avionics Industry. In *33rd Digital Avionics Systems Conference (DASC)*, 2014.

[91] Internet Engineering Task Force (IETF). RFC 1155: Structure and Identification of Management Information for TCP/IP-based internets. Technical report, 1990. URL http://datatracker.ietf.org/doc/rfc1155/.

[92] Internet Engineering Task Force (IETF). RFC 6020 - YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF).

[93] Lukasz Wisniewski, Satendrasingh Chahar, and Juergen Jasperneite. Seamless reconfiguration of time triggered ethernet based protocols. In *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, 2015.

[94] IEEE 802.1AB-2009: Standard for Local and metropolitan area networks Station and Media Access Control Connectivity Discovery. Technical report, 2005.

[95] Telecommunications Industry Association (TIA). Link Level Discovery Protocol - Media Endpoint Discovery (LLDP-MED). Technical report, 2005.

[96] Farzaneh Pakzad, Marius Portmann, Wee Lum Tan, and Jadwiga Indulska. Efficient Topology Discovery in Software Defined Networks. In *Signal Processing and Communication Systems (ICSPCS)*, 2014.

[97] Organisation Internationale de Normalisation. International Standard ISO / IEC Telecommunications and information. (ISO 8473), 2012.

[98] Leonardo Ochoa-Aday, Cristina Cervelló-Pastor, and Adriana Fernández-Fernández. Discovering the Network Topology: An Efficient Approach for SDN. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 2016.

[99] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Enabling Fast Failure Recovery in OpenFlow Networks. *IEEE Design of Reliable Communication Networks*, 2011.

[100] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. In-Band Control, Queuing, and Failure Recovery Functionalities for OpenFlow. *IEEE Network*, 30(1), 2016.

[101] IEEE. IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks. Technical report, 2010.

[102] Roland Kammerer and Roman Obermaisser. Dynamic configuration of a time-triggered router for controller area network. In *Emerging Technologies Factory Automation (ETFA)*, 2012.

[103] Morteza Hashemi Farzaneh and Alois Knoll. An Ontology-based Plug-and-Play Approach for In-Vehicle Time-Sensitive Networking (TSN). In *Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2016.

[104] Internet Engineering Task Force (IETF). RFC 1531 - Dynamic Host Configuration Protocol. 1993.

[105] Internet Engineering Task Force (IETF). RFC 3927 - Dynamic Configuration of IPv4 Link-Local Addresses. 2005.

[106] Internet Engineering Task Force (IETF). IPv6 Stateless Address Autoconfiguration (RFC 2462). Technical report, IETF, 1998.

[107] Lars Dürkop, Henning Trsek, Jürgen Jasperneite, and Lukasz Wisniewski. Towards Autoconfiguration of Industrial Automation Systems: A Case Study Using Profinet IO. In *17th Conference on Emerging Technologies Factory Automation (ETFA)*, 2012.

[108] Lars Dürkop, Jahanzaib Imtiaz, Henning Trsek, Lukasz Wisniewski, and Jürgen Jasperneite. Using OPC-UA for the autoconfiguration of real-time Ethernet systems. In *International Conference on Industrial Informatics (INDIN)*, 2013.

[109] Gunther Reinhart, Stefan Krug, Stefan Hüttner, Z. Mari, F. Riedelbauch, and M. Schlögel. Automatic configuration (Plug & Produce) of Industrial Ethernet networks. In *9th IEEE/IAS International Conference on Industry Applications (INDUSCON 10)*, 2010.

[110] Jahanzaib Imtiaz, Jürgen Jasperneite, Karl Weber, Franz Josef Götz, and Gunnar Lessmann. A novel method for Auto configuration of Realtime Ethernet Networks. In *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2008.

[111] Peter Heise, Fabien Geyer, Alexandros Elefsiniotis, and Roman Obermaisser. Towards a Deterministic Mixed Criticality High Availability Seamless Redundancy (HSR) Ring. In *8th International Workshop on Communication Technologies for Vehicles (Nets4Cars/Nets4Trains/Nets4Aircraft)*, 2015.

[112] Internet Engineering Task Force (IETF). RFC 1242 - Benchmarking Terminology for Network Interconnection Devices. 1991.

[113] OMNeT++. URL http://www.omnetpp.org/.

[114] INET Framework. URL http://inet.omnetpp.org/.

[115] Peter Heise, Fabien Geyer, and Roman Obermaisser. Deterministic OpenFlow: Performance Evaluation of SDN Hardware for Avionic Networks. In *2nd International Workshop on Management of SDN and NFV Systems at 11th International Conference on Network and Service Management (CNSM)*, 2015.

[116] Ericsson Innovation Center in Brazil in a partnership with CPqD in technical collaboration with Ericsson Research. CPqD OpenFlow 1.3 Software Switch. 2015. URL https://github.com/CPqD/ofsoftswitch13.

[117] Peter Heise, Fabien Geyer, and Roman Obermaisser. TSimNet: An industrial Time Sensitive Networking simulation framework based on OMNeT++. In *8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2016.

[118] AVnu. AVB Automotive Profile. URL https://standards.ieee.org/events/automotive/2014/17_Automotive_E-AVB_Profile.pdf.

[119] MEF Forum. MEF 10.3 - Ethernet Services Attributes. URL https://www.mef.net/Assets/Technical_Specifications/PDF/MEF_10.3.pdf.

[120] Peter Heise, Marc Lasch, Fabien Geyer, and Roman Obermaisser. Self-Configuring Real-Time Communication Network based on OpenFlow. In *22nd IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2016.

[121] Internet Engineering Task Force (IETF). RFC 2608 - Service Location Protocol, Version 2. 1999.

[122] Roman Obermaisser and Ayman Murshed. Incremental, Distributed, and Concurrent Scheduling in Systems-of-Systems with Real-Time Requirements. *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 2015.

[123] Ryu OpenFlow Controller. URL https://osrg.github.io/ryu.

[124] Mininet. URL http://mininet.org/.

[125] Peter Heise, Fabien Geyer, and Roman Obermaisser. Self-Configuring Deterministic Network with In-Band Configuration Channel. In *1st International Workshop on Software Defined Networks and Network Function Virtualization (SDN-NFV) at Fourth IEEE International Conference on Software Defined Systems (SDS-2017)*, 2017.

[126] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. TCP Network Calculus: The case of large delay-bandwidth product. In *21st Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.

[127] Open vSwitch. URL http://openvswitch.org/.

[128] Jarno Rajahalme. [ovs-dev] [PATCH v3 0/4] Meter implementation for userspace datapath. URL https://mail.openvswitch.org/pipermail/ovs-dev/2015-November/305971.html.

[129] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and Others. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 2014.