Time Sensitive Networking for Virtualized Integrated Real-Time Systems

DISSERTATION
To Obtain the Degree of Doctor of Engineering

Submitted By
Maryam Pahlevan

Submitted To
Faculty of Elektrotechnik und Informatik
University of Siegen
Siegen 2019

Supervisor And First Appraiser
Prof. Dr. Roman Obermaisser
University of Siegen

Second Appraiser
Prof. Dr. Nicolas Navet
University of Luxembourg

Date of the Oral Examination
29. January 2020

Abstract

For many years, embedded computer systems have been designed based on federated architectures where every service of a system is mapped to a dedicated hardware unit. A federated system architecture, despite several advantages such as low architectural complexity and high level of dependability, requires a new dedicated node for every newly added function. Therefore, integrated system architectures were introduced to address the scalability issues of federated systems, assuming that every node comprises several isolated partitions and different services can execute on a single node. Nevertheless, the integrated systems require many modifications in case of any changes in a system which lead to the costly design, integration, verification and maintenance processes. Thereby, a domain-specific integrated system can be transformed into a more generic solution through resource virtualization. Such a virtualized integrated system can host several functions with different reliability constraints and temporal requirements on the virtualized computing resources. The key advantage of the virtualized integrated systems is that they are reconfigurable easily and at very low cost. Each function in the virtualized integrated system may reside either on different partitions of a single processing node or on different computers and communicate with other functions via a common networking infrastructure. Therefore, the non-functional requirements of the virtualized integrated system, including reliability, availability, integrity, safety and maintainability highly depend on the capabilities of the communication infrastructure. Moreover, the networking technology of the integrated system has a significant impact on the design, complexity management and integration process.

The Ethernet standard due to its widespread usage is considered as a promising networking solution for the virtualized integrated system. However, Ethernet does not offer the fault-tolerant and deterministic communication infrastructure that is essential for modern embedded systems. Therefore, this thesis introduces a communication layer of a virtualized integrated system based on the principles of the Time-Sensitive Networking (TSN) standard, which encompasses a series of protocol extensions to the Ethernet standard. TSN offers real-time capability and performance improvements while benefiting from high bandwidth and seamless connectivity of Ethernet technologies.

It is essential to verify the correctness and applicability of TSN mechanisms as a networking solution for the virtualized integrated system. Therefore, this thesis presents a simulation framework for the TSN standard, which is developed as a multi-hop switched Ethernet network. The simulation framework is generic with stable interfaces between simulation components. Therefore, it can be universally applied to simulate different applications and to gain insights into different architectural decisions (e.g. different topologies, different redundancy degrees). In addition, it can be extended using additional sub-protocols of TSN and modified to incorporate future changes introduced by the TSN working group. The TSN simulator also includes dynamic configuration services, thereby enabling the modelling of dynamic applications (like train inauguration) and system adaptation (e.g. fault

recovery). Furthermore, there is neither an existing simulation framework nor an actual TSN device which contains different TSN features such as remote configuration, clock synchronization and time-aware shaping simultaneously. Hence, the presented TSN simulator provides a comprehensive simulation platform for modelling, performance and reliability evaluation of TSN networks. The empirical results based on a real-world use case illustrate that the central configuration model of TSN enables the remote management and configuration of the emulated network. Moreover, according to the experimental results, the simulation models with TSN features satisfy the stringent timing and reliability requirements of the virtualized integrated system.

TSN offers determinism using Time-Triggered (TT) transmission schedules which are expressed as Gate Control Lists (GCL). The scheduling problem arising from the GCL synthesis is NP-complete. Moreover, the feasibility of running real-time applications over different virtualized computing resources makes the search space of legitimate schedules even bigger. Therefore, the optimization algorithms for the search space exploration are a vital element for the deployment of TSN. This thesis presents a fast Genetic Algorithm (GA) and a Heuristic List Scheduler (HLS) which are designed to compute GCLs by addressing the interdependence of routing and scheduling constraints. The primary goal of these schedulers is to satisfy the deadlines of real-time applications while optimizing the TT transmission makespan and the overhead of TT communication. The experimental results show that GA and HLS improve the transmission makespan on average by 31 % and 39 % respectively compared to an existing scheduling strategy which uses fixed routing. Moreover, in experiments, it is observed that the schedulability ratios of GA and HLS significantly increase (on average by 71 % and 73 % respectively) compared to a two-step scheduler.

The seamless recovery from faulty behaviours is vital for many modern embedded systems, since failures in such systems may result in irreparable environmental damages and substantial financial losses. Therefore, this thesis extends the scheduling strategies described above to support the TSN redundancy management mechanism. To this end, the message replication, elimination of replicas and the redundant path selection are incorporated to the TSN schedulers. The main goal of the fault-tolerant TSN schedulers is to optimize the overall system reliability based on application and platform models while satisfying the real-time constraints of the application. The system reliability considers the redundancy in the application models (e.g. redundant and non-redundant real-time jobs), the redundancy in the platform models and the reliability of the TSN platform components (e.g. end systems, switches and links) and novel TSN-based fault-tolerance mechanisms such as IEEE 802.1CB. The empirical results show that the fault-tolerant TSN schedulers enhance the system reliability of the schedules compared to TSN schedulers without fault-tolerance mechanism at the expense of an increased makespan.

Zusammenfassung

Embedded-Computer-Systeme werden Seit vielen Jahren entwickelt. Sie basieren auf föderierten Architekturen, bei denen jeder Dienst eines Systems auf eine dedizierte Hardwareeinheit abgebildet wird. Eine föderierte Systemarchitektur erfordert trotz mehrerer Vorteile wie geringe Architekturkomplexität und hohe Zuverlässigkeit für jede neu hinzugefügte Funktion einen neuen dedizierten Knoten. Daher wurden integrierte Systemarchitekturen eingeführt, um die Skalierbarkeitsprobleme von föderierten Systemen zu lösen, vorausgesetzt, dass jeder Knoten aus mehreren isolierten Partitionen besteht und verschiedene Dienste auf einem einzigen Knoten ausgeführt werden können. Dennoch erfordern die integrierten Systeme Modifikationen, wenn sich in einem System Änderungen ergeben, die zu kostspieligen Design-, Integrations-, Verifikations- und Wartungsprozessen führen. Dadurch kann ein domänenspezifisches integriertes System durch Ressourcenvirtualisierung in eine generische Lösung umgewandelt werden. Ein solches virtualisiertes integriertes System kann mehrere Funktionen mit unterschiedlichen Zuverlässigkeitseinschränkungen und zeitlichen Anforderungen an die virtualisierten Computerressourcen verwalten. Der ausschlaggebende vorteil der virtualisierten integrierten Systeme besteht darin, dass sie einfach und kostengünstig rekonfigurierbar sind. Jede Funktion im virtualisierten integrierten System kann sich entweder auf verschiedenen Partitionen eines einzelnen Verarbeitungsknotens oder auf verschiedenen Computern befinden und über eine gemeinsame Netzwerkinfrastruktur mit anderen Funktionen kommunizieren. Daher hängen die nicht-funktionalen Anforderungen an das virtualisierte integrierte System, einschließlich Zuverlässigkeit, Verfügbarkeit, Integrität, Sicherheit und Wartbarkeit, stark von den Fähigkeiten der Kommunikationsinfrastruktur ab. Darüber hinaus hat die Netzwerktechnologie des integrierten Systems einen wesentlichen Einfluss auf das Design, das Komplexitätsmanagement und den Integrationsprozess.

Der Ethernet-Standard gilt aufgrund seiner weiten Verbreitung als vielversprechende Netzwerklösung für das virtualisierte Gesamtsystem. Ethernet bietet jedoch nicht die fehlertolerante und deterministische Kommunikationsinfrastruktur, die für moderne Embedded-Systeme unerlässlich ist. Daher wird in dieser Arbeit eine Kommunikationsschicht eines virtualisierten integrierten Systems vorgestellt, das auf den Prinzipien des Time-Sensitive Networking (TSN) Standards basiert, der eine Reihe von Protokollerweiterungen zum Ethernet-Standard umfasst. TSN bietet Echtzeitfähigkeit und Leistungssteigerung und profitiert gleichzeitig von der hohen Bandbreite und der nahtlosen verbindung von Ethernet-Technologien.

Es ist unerlässlich, die Richtigkeit und Anwendbarkeit von TSN-Mechanismen als Netzwerklösung für das virtualisierte integrierte System zu überprüfen. Daher stellt diese Arbeit einen Simulationsrahmen für den TSN-Standard dar, der als Multi-Hop-Switched-Ethernet-Netzwerk entwickelt wurde. Das Simulationsframework ist generisch mit stabilen Schnittstellen zwischen den Simulationskomponenten. Daher kann es universell eingesetzt werden, um verschiedene Anwendungen zu simulieren und Einblicke in verschiedene Architekturentscheidungen

(z.B. unterschiedliche Topologien, unterschiedliche Redundanzgrade etc) zu gewinnen. Darüber hinaus kann es durch zusätzliche Unterprotokolle des TSN erweitert und an zukünftige Änderungen der TSN-Arbeitsgruppe angepasst werden. Der TSN-Simulator beinhaltet auch dynamische Konfigurationsdienste, die die Modellierung dynamischer Anwendungen (z.B. Zuginbetriebnahme) und Systemanpassungen (z.B. Fehlerbehebung) ermöglichen. Darüber hinaus gibt es weder ein bestehendes Simulationsframework noch ein aktuelles TSN-Gerät, das verschiedene TSN-Features wie Fernkonfiguration, Uhrensynchronisation und zeitgesteuerte Formgebung gleichzeitig enthält. Damit bietet der vorgestellte TSN-Simulator eine umfassende Simulationsplattform zur Modellierung, Leistungs- und Zuverlässigkeitsbewertung von TSN-Netzen. Die empirischen Ergebnisse, die auf einem realen Anwendungsfall basieren, zeigen, dass das zentrale Konfigurationsmodell von TSN die Fernverwaltung und Konfiguration des emulierten Netzwerks ermöglicht. Darüber hinaus erfüllen die Simulationsmodelle mit TSN-Features nach den experimentellen Ergebnissen die hohen Anforderungen an Timing und Zuverlässigkeit des virtualisierten Gesamtsystems.

TSN bietet Determinismus mit Time-Triggered (TT) Übertragungsplänen, die als Gate Control Lists (GCL) ausgedrückt werden. Das Planungsproblem, das sich aus der GCL-Synthese ergibt, ist NP-complete. Darüber hinaus macht die Möglichkeit, Echtzeitanwendungen über verschiedene virtualisierte Computerressourcen auszuführen, den Suchraum für legitime Zeitpläne noch größer. Daher sind die Optimierungsalgorithmen für die Suchraumerkundung ein wesentliches Element für den Einsatz von TSN. Diese Arbeit stellt einen schnellen genetischen Algorithmus (GA) und einen heuristischen Listenplaner (HLS) vor, die dazu bestimmt sind, GCLs zu berechnen, indem sie die Wechselwirkung von Routing und Scheduling-Beschränkungen berücksichtigen. Das Hauptziel des Planer ist es, die Fristen von Echtzeitanwendungen einzuhalten und gleichzeitig die TT-Übertragungszeit und den Aufwand der TT-Kommunikation zu optimieren. Die experimentellen Ergebnisse zeigen, dass GA und HLS die Übertragungsspanne im Durchschnitt um 31 % bzw. 39 % verbessern, verglichen mit einer bestehenden Planungsstrategie, die ein festes Routing verwendet. Darüber hinaus wird in Experimenten beobachtet, dass die Planbarkeitsverhältnisse von GA und HLS im Vergleich zu einem zweistufigen Planer signifikant steigen (durchschnittlich um 71 % bzw. 73 %).

Die nahtlose Wiederherstellung von Fehlverhalten ist für viele moderne eingebettete Systeme von entscheidender Bedeutung, da Ausfälle in solchen Systemen zu irreparablen Umweltschäden und erheblichen finanziellen Verlusten führen können. Daher erweitert diese Arbeit die oben beschriebenen Planungsstrategien, um den TSN Redundanzmanagementmechanismus zu unterstützen. Zu diesem Zweck werden die Nachrichtenreplikation, die Eliminierung von Replikaten und die redundante Pfadauswahl in die TSN-Scheduler integriert. Das Hauptziel der fehlertoleranten TSN-Scheduler ist es, die Gesamtsystemzuverlässigkeit basierend auf Anwendungs- und Plattformmodellen zu optimieren und gleichzeitig die Echtzeitbeschränkungen der Anwendung zu erfüllen. Die Systemzuverlässigkeit berücksichtigt die Redundanz in den Anwendungsmodellen (z.B. redundante und nicht-redundante Echtzeitaufträge), die Redundanz in den Plattformmodellen und die Zuverlässigkeit der TSN-Plattformkomponenten (z.B. Endsysteme, Switches und Links) und neuartige TSN-basierte Fehlertoleranzmechanismen wie IEEE 802.1CB. Die empirischen Ergebnisse zeigen, dass die fehlertoleranten TSN-Scheduler die Systemzuverlässigkeit der Scheduler im Vergleich zu TSN-Schedulern ohne Fehlertoleranzmechanismus auf Kosten einer erhöhten Makepan verbessern.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my doctoral supervisor Prof. Dr Obermaisser for his support throughout my PhD study. I am very grateful for his dedication, exciting ideas, and funding that facilitate my PhD experience. If it was not for his motivation, and immense knowledge, writing of this thesis would not be possible. My special gratitude goes out to members of the chair for Embedded System at University of Siegen for accompanying me through this tough journey. In particular, I would like to thank Manuela for helping me with paperwork. I am especially grateful for my office mates: Tobias, Hongji and Daniel, who put up with my tantrums over the last three years. I also want to thank my students Jonas and Balakrishna, who have contributed to the development of the simulation framework as their master theses. Their works were very instrumental to this PhD thesis.

I want to thank my friends Nadra, Adele, Setareh and Sara, who provided me with the generous technical and emotional support during this time. I would also like to thank Tobias for making a housing search in Siegen such an effortless and pleasant experience. Special thanks to Fariba and Ehsan for hosting me frequently over weekends and being my only family in Germany.

And lastly, I would like to extend my gratitude to all my family members for their endless love and continuous encouragement during my PhD study. I dedicated this thesis to my parents, who diligently support me through all my ups and downs. For my Siblings Sara, Mahdi and Mohadeseh, who inspire me to strive for academic excellence.

I would like to acknowledge the financial support from the European project-Safe4Rail, which provides an opportunity for my PhD study and related research works.

Contents

List of Figures

List of Tables

List of Abbreviations

GA	**G**enetic **A**lgorithm
HLS	**H**euristic **L**ist **S**cheduler
ECU	**E**Lectronic **C**ontrol **U**nit
DAS	**D**istributed **A**pplication **S**ubsystem
IMA	**I**ntegrated **M**odular **A**vionics
CPU	**C**ontrol **P**rocessing **U**nit
IoT	**I**nternet **o**f **T**hings
TT	**T**ime **T**riggered
RC	**R**ate **C**onstrained
BE	**B**est **E**ffort
TSN	**T**ime **S**ensitive **N**etworking
GCL	**G**ate **C**ontrol **L**ist
V/V	**V**erification **a**nd **V**alidation
TAS	**T**ime **A**ware **S**aper
FRER	**F**rame **R**eplication **a**nd **E**limination **f**or **R**eliability
BMCA	**B**est **M**aster **C**lock **A**lgorithm
ADAS	**A**dvanced **D**river **A**ssistance **S**ystem
GPS	**G**lobal **P**ositioning **S**ystem
FCR	**F**ault **C**ontainment **R**egion
LAN	**L**ocal **A**ccess **N**etwork
WAN	**W**ide **A**ccess **N**etwork
DEC	**D**igital **E**quipment **C**orporate
MAC	**M**ultiple **A**ccess **C**ontrol
FCS	**F**rame **C**heck **S**equence
QoS	**Q**uality **o**f **S**ervice
VLAN	**V**irtual **LAN**
RSTP	**R**apid **S**panning **T**ree **P**rotocol
MRP	**M**edia **R**edundancy **P**rotocol
PRP	**P**arallel **R**edundancy **P**rotocol
HSP	**H**igh-availability **S**eamless **P**rotocol
CRP	**C**ross-network **R**edundancy **P**rotocol
BRP	**B**eacon **R**edundancy **P**rotocol
DRP	**D**istributed **R**edundancy **P**rotocol
RRP	**R**ing-based **R**edundancy **P**rotocol
STP	**S**panning **T**ree **P**rotocol
SSP	**S**ingle **S**panning **T**ree
CST	**C**ommon**S**panning **T**ree
MRM	**M**edia **R**edundancy **M**anager
MRC	**M**edia **R**edundancy **C**leint
DAN	**D**oubly **A**ttached **N**ode
SAN	**S**ingle **A**ttached **N**ode
RCT	**R**edundancy **C**ontrol **T**railer
LSDU	**L**ist **S**ervice **D**ata **U**nit

PTP	**P**recision **T**ime **P**rotocol
PLC	**P**rogrammable **L**ogic **C**ontroller
NTP	**N**etwork **T**ime **P**rotocol
OSI	**O**pen **S**ystem **I**nterconnection
MII	**M**edium **I**ndependent **I**nterface
gPTP	**g**eneralized **P**recision **T**ime **P**rotocol
EPON	**E**thernet **P**assive **O**ptical **N**etwork
CSN	**C**oordinated **S**hared **N**etwork
NRR	**N**eighbour **R**ate **R**atio
CSRO	**C**umulative **S**caled **R**ate **O**ffset
TLV	**T**ype **L**ength **V**alue
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
VL	**V**irtual **L**ink
BAG	**B**andwidth **A**llocation **G**ap
PCF	**P**rotocol **C**ontrol **F**rame
SM	**S**ynchronization **M**aster
CM	**C**ompression **M**aster
IF	**I**tegration **F**rame
AVB	**A**udio **V**ideo **B**ridging
SRP	**S**tream **R**eservation **P**rotocol
MMRP	**M**ultiple **M**AC **R**egistration **P**rotocol
MVRP	**M**ultiple **V**LAN **R**egistration **P**rotocol
MSRP	**M**ultiple **S**tream **R**egistration **P**rotocol
SR	**S**tream **R**eservation
CoS	**C**lass **o**f **S**ervice
CBS	**C**redit **B**ased **S**haping
PCP	**P**riority **C**ode **P**oint
ACL	**A**ccess **C**ontrol **L**ist
MTU	**M**aximum **T**ransmission **U**nit
UNI	**U**ser **N**etwork **I**nterface
CNC	**C**entralized **N**etwork **C**onfiguration
CUC	**C**entralized **U**ser **C**onfiguration
YANG	**Y**et **A**nother **N**ext **G**eneration
URI	**U**niform **R**esource **I**dentifier
RPC	**R**emote **P**rocedure **C**all
SMT	**S**atisfiability **M**odulo **T**heory
MIP	**M**ixed **I**teger **P**rogramming
OMT	**O**ptimisation **M**odulo **T**heory
GRASP	**G**reedy **R**andomized **A**daptive **S**earch **P**rocedure
PB	**P**seudo **F**oolean
LCM	**L**east **C**ommon **M**ultiple
LS	**L**ist **S**cheduler
EA	**E**volutionary **A**lgorithm
MTTF	**M**ean **T**ime **T**o **F**ailure
CCU	**C**entral **C**omputing **U**nit
HMI	**H**uman **M**achine **I**nterface
FTGA	**F**ault **T**olernat**G**enetic **A**lgorithm
FTHLS	**F**ault **T**olernat**H**euristic **L**ist
ATS	**A**synchronous **T**raffic **S**haper
ABS	**A**daptive **B**andwidth **S**haring
ASW	**A**daptive **S**lot **W**indow

PTA	**P**riced **T**imed **A**utomata
SEU	**S**ingle **E**vent **U**pset
SDN	**S**oftware **D**efined **N**etwork
GUI	**G**raphical **U**ser **I**nterface
FSM	**F**inite **S**tate **M**achine
ICI	**I**nterface **C**ontrol **I**nformation
LLC	**L**ogical **L**ink **C**ontrol
DLL	**D**ata **L**ink **L**ayer
BPDU	**B**ridge **P**rotocol **D**ata **U**nit
MSTP	**M**ultiple **S**panning **T**ree **P**rotocol
ETB	**E**thernet **T**rain **B**ackbone
ECN	**E**thernet **C**onsist **N**etwork
DSCP	**D**ifferentiated **S**ervices **C**ode **P**oint

To my parents

Chapter 1

Introduction

During the last decades, embedded computer systems were used extensively for the execution of control and measurement applications in different domains like automotive, avionics and industrial automation. For instance, in the automotive domain, an Electronic Control Unit (ECU) can substitutes a hydraulic control system and also provide an opportunity to add new services like advanced driver-assistance [1]. For many years, a domain-specific embedded system was defined based on a federated architecture, assuming that there is a one-to-one mapping between every service of a system and a hardware unit. The federated system architecture offers low architectural complexity and a high level of dependability which are significant concerns of safety-critical systems, but it requires a new dedicated node for every newly added function. Due to this design principle, the number of hardware units increases considerably to enhance the fault-tolerance of the system [2].

To resolve the scalability issue of the federated system architecture and also to support the ever-increasing number of new functionalities within an embedded system, the integrated distributed system architecture was introduced [3]. In this architecture, unlike the federated architecture, every node comprises several isolated partitions. Consequently, different services can execute on a single node. This system architecture integrates several autonomous Distributed Application Subsystems (DAS) with different levels of criticality. The main goal of the integrated system architecture is to dedicate a separate partition of the node to every function of the DAS. Therefore, this system architecture results in a more compact and cost-efficient embedded system structure by decreasing the number of hardware units. Nevertheless, the integration of different functions increases the complexity of the overall system. The integrated system architecture aims to achieve the same level of reliability and composability as the federated system architecture using a robust synchronization mechanism, the deterministic communication infrastructure and solid diagnosis services [2].

Integrated Modular Avionics (IMA) [4] and AUTOSAR [5] are well-known examples of integrated distributed system architectures in the field of avionic and automotive systems, respectively. Each of these integrated distributed systems comprises a computing node with a partitioned operating system and a powerful Central Processing Unit (CPU). These systems allocate an operating system partition to every function within a specific DAS. Each partition of the operating system is protected from the run-time interference and fault propagation of other partitions. The key objective of the mentioned integrated systems is to provide reliable mission-critical services at reduced cost. To achieve a cost-efficient integrated embedded system, the existing stable components, either hardware units or software components, are reused to the greatest extent possible. Besides, the cost of the integrated system is reduced significantly by migrating the services from several nodes with lower computational power to a few powerful computing nodes [1].

1.1 Virtualized Integrated Systems

In the last years, a system designer generally determined the embedded system architecture and integration solutions of a specific problem, considering the resource constraints, temporal and safety requirements. This type of problem-specific solutions did not apply to other industrial use-cases. Moreover, these solutions required many modifications in case of any changes in a system like reconfiguration. This approach leads to a costly design, integration, verification and maintenance processes. A problem-specific system architecture can be transformed into a more generic solution by using open technologies. The key advantage of the open technologies is that they are easily reconfigurable and provide lower cost. Cloud computing, as an example of the open technologies, fits best to the requirements of the generic integrated system architecture. However, it does not meet the stringent temporal and dependability constraints of mission-critical applications.

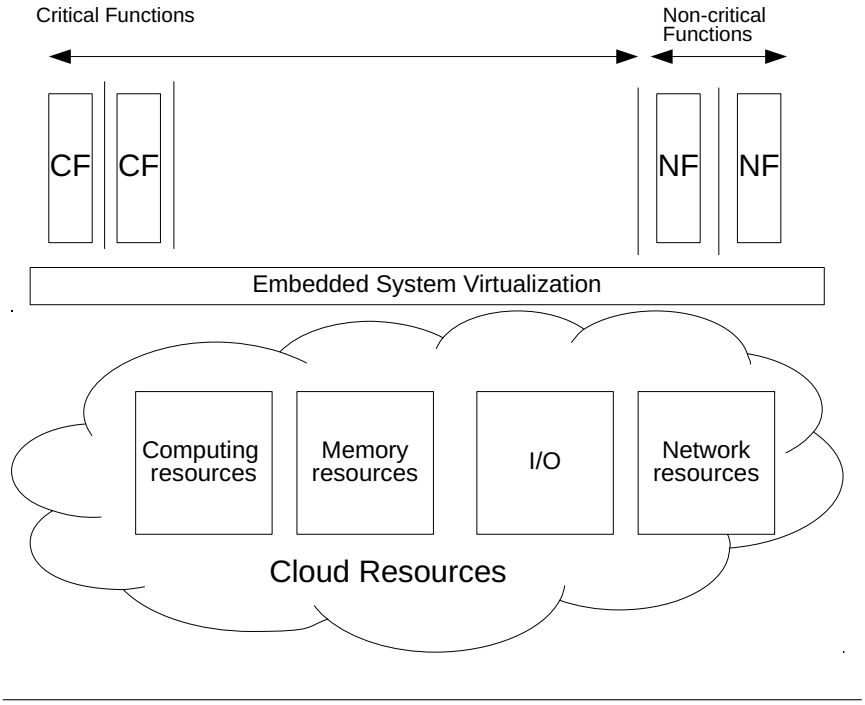


FIGURE 1.1: Virtualized integrated system architecture [6]

The virtualized integrated system architecture was introduced as a baseline for modern embedded systems. This architecture considers the virtualized resources for the applications with different levels of criticality. To be more specific, the distributed system with the virtualized integrated architecture allocates the cloud of virtualized resources to a wide range of functions. Each function may have different reliability constraints (e.g. fail-operational and fail-safe) and timing requirements (e.g. hard real-time, soft real-time and best effort). Such cloud-based embedded systems are capable of configuring any function either critical or non-critical to access the necessary computing resources via a common networking infrastructure. A critical function must have exclusive access to shared resources in specified time intervals. However, this exclusive access should not result in the resource starvation of non-critical functionalities. The deployment of a function over a cloud of computational and networking resources eliminates the spatial dependencies. Thus it enables a flat architecture in addition to a hierarchical structure. Furthermore, the

virtualized integrated system requires a limited number of modifications to support any domain-specific use-case with a specific application structure and properties. This feature also makes an incremental and modular certification process feasible. It also results in cost and time-efficient verification and maintenance activities [7]. Figure 1.1 presents the simplified structure of the virtualized integrated system.

SCARLETT [8] and ASHLEY [9] are two examples of European aerospace projects which are developed based on the concept of the virtualized integrated system architecture. These projects aimed to be compatible with ARINC664 protocols [10] while meeting the European aerospace industry requirements. The integration solutions that were proposed in these projects share similar challenges as the generic integrated system architecture in other domains like the next-generation IMA and Internet-of-Things (IoT) systems that are tailored for mission-critical applications. The significant challenges of such virtualized embedded systems are the recurring configuration changes, high complexity of resource management due to the shared virtualized resources, modular certification and distributed real-time applications.

In the development of virtualized embedded systems, the primary driver is to provide a highly reliable and reconfigurable distributed integrated platform. This platform can host and integrate safety-critical functions on a large number of computational and communication resources. Moreover, a virtualized integrated system architecture enables decoupling of the software components and controlled hardware units (e.g. actuators and sensors)[6].

The system-level requirements of the virtualized integrated system architecture can be summarized as follows:

- The key objective of the virtualized integrated system is to optimize the time and cost of the design, implementation, verification, deployment, upgrade, reconfiguration and maintenance activities. For this purpose, the different critical and non-critical functions are integrated on common communication and computational resources.

- The virtualized integrated system architecture provides a highly-dependable platform for the mission-critical applications. This aspect is crucial for the majority of domain-specific use-cases, because the failure of the mission-critical applications can lead to irreparable environmental damage and huge financial losses.

- The virtualized integrated platform should also be fault-tolerant. Since the hardware units and software components experience different faults during their life-time and these faults could potentially, result in catastrophic outcomes for the mission-critical applications (e.g. braking system in a car or train). The reason is that the mission-critical applications must continue to operate even in the presence of failures.

- In a virtualized integrated platform, the functions with the highest criticality level (e.g. braking control subsystem) can be integrated with the non-critical functions (e.g. entertainment subsystem) without any degradation of performance, safety and timing requirements.

- The virtualized integrated platform can be verified and certified independently. This means that the verification and certification of the platform can be done without considering the integrated functions. In addition, this platform enables the incremental verification and certification, meaning that for each new function, the testing of the existing integrated functions is not required.

- The virtualized integrated platform must be configured so that the strict timing and reliability requirements of the mission-critical application are met.
- The configuration management system of the virtualized integrated platform must handle the initial deployment and also the reconfiguration of the software components and hardware units in a reliable and cost-efficient manner.
- The virtualized integrated platform provides interoperability between different vendor-specific hardware units and software components. This implies that every function can be allocated to different vendor-specific resources and can communicate with other functions seamlessly. [6].

1.2 Communication Layer of the Virtualized Integrated System

As stated above, in a virtualized integrated system, different functions, either critical or non-critical, can be hosted anywhere on the virtualized computing resources. This implies that each function can reside either on different partitions of a single processing node or on different computers. Consequently, for communication between different functions, a deterministic network is required. The networking infrastructure plays a key role in the virtualized integrated system due to its distributed nature. In addition, the non-functional requirements of the virtualized integrated system, including reliability, availability, integrity, safety and maintainability highly depend on the capabilities of the communication infrastructure. Moreover, the networking technology of the integrated system has a major impact on the design, complexity management and integration process.

Different types of traffic can be exchanged over the network in the virtualized integrated system. Every traffic class can have its own temporal and safety requirements. Hence, the network must guarantee deterministic and reliable communication services for the mission-critical functions while it serves less critical functions. For this purpose, the network of the virtualized integrated system uses temporal and spatial partitioning of the communication resources. [6]

The network of the integrated system should offer the following services:

- Services regarding transportation of Time-Triggered (TT) streams, Rate-Constrained (RC) traffic and best-effort messages
- Services regarding configuration and reconfiguration of network components like end systems and switches
- Services regarding clock synchronization like distribution of timing information
- Services regarding fault prevention, detection and recovery

The communication infrastructure handles each traffic type differently. For instance, it transmits the TT messages which have strict timing constraints at the predefined cycles. For TT communication, Ethernet-based technologies are widely deployed in different industrial systems like automotive, avionics and industrial automation. In recent years, the Time-Sensitive Networking (TSN) task group [11] introduced a novel time-aware shaping concept [12] for data transmission with different criticality levels. The TTEthernet and TSN standards as examples of Ethernet-based technologies, offer determinism in terms of jitter and end-to-end delay for TT

communication, but they have subtle differences in their mechanisms that may impact the design of the distributed integrated systems. The other types of traffic like RC messages are transmitted when no TT message is scheduled since the network of the integrated system offers a deterministic message delivery for the time-critical functions. The RC streams are commonly dispatched based on either the shaping mechanism defined in AFDX [10] or TSN credit-based shaper [13].

The virtualized integrated system uses virtualized computing and networking resources for different functions. Therefore, unlike the existing static industrial platforms in cars and aeroplanes, the structure of the virtualized integrated system can frequently change due to newly added functionalities. The network should adapt to modifications via reconfiguration. For configuration and reconfiguration of different network components, the network should provide a service which collects the information related to system changes, compute the new configuration data and then disseminates this information to the network components.

The network of the integrated system should provide a robust synchronization service, because for deterministic communication, the network devices must share the same notion of time. However, the local clocks in the majority of network protocols are not synchronized with each other. Hence, a clock synchronization service aims to synchronize the local clocks of all network devices to a global time base. To achieve this goal, the clock synchronization service enables a fault-tolerant and reliable dissemination of timing information as well as the computation of clock correction terms.

The network should also offer health monitoring services. These services aim to firstly detect the faulty behaviours and then execute the appropriate recovery mechanisms based on the type of fault [14].

In addition to the services as mentioned earlier, the network resources in the virtualized integrated system can be managed in the following ways:

- Bandwidth virtualization: to simulate individual resources for data communication of critical and non-critical functions
- Topology virtualization: to simulate different network structures over the same topology. To achieve this goal, the network applies different policing mechanisms so that each network device have access to some parts of the network while it does not have access to other parts of the network.
- Configuration virtualization: to simulate a common memory for sharing the global configuration parameters [6].

1.3 Thesis Objectives

This thesis focuses on the modelling and simulation of the communication layer of a virtualized integrated system. As mentioned before, the virtualized integrated system is designed to host a wide range of critical and non-critical functions on a single platform. In the virtualized integrated system, the communication technologies should be selected considering the networking capabilities and requirements of the applications with different criticality levels. In addition, the networking infrastructure is intended to require less cabling, robust clock synchronization, reliable transmission of mixed-criticality traffic, simplified system integration and configuration management.

Ethernet fulfills different needs of various stakeholders from the high bandwidth demand to the seamless connectivity between vendor-specific devices. Due to widespread usage and massive success of Ethernet technologies, Ethernet is considered as a promising solution for the industrial and deterministic networks. Temporal properties and dependability requirements play a vital role in the development of emerging mission-critical applications and improvements of current technologies (e.g. industrial automation). However, the Ethernet standard is not designed to provide deterministic behaviours which are essential for real-time applications [15]. In conventional Ethernet-based networks, time is only used as a component for performance measurements, not as a correctness metric [16]. Therefore, several extensions of Ethernet were introduced to offer determinism such as bounded end-to-end delay and low jitter.

The most recent real-time capable Ethernet extension is Time-Sensitive Networking (TSN). The convergence of synchronous, asynchronous and best-effort traffic on a single network is the key aspect of TSN. Furthermore, TSN introduced a series of standards that offer high dependability, fault-tolerant clock synchronization and performance improvements. It is necessary to evaluate and validate the applicability of TSN solutions for the network of the virtualized integrated system. Networking experts and technology manufacturers use simulation frameworks extensively to emulate and validate new networking solutions, since the implementation and deployment of novel protocols on hardware is a very time consuming and expensive process. Besides, during the development phase, the networking protocols usually require plenty of modifications due to constant changes in the solution designs. As many TSN protocols like IEEE 802.1As-Rev [17] are still in the development process and not finalized yet, the simulation models are developed for the time-based and non-time-based features of TSN. The TSN simulator introduced in this thesis provides an opportunity to validate correctness and applicability of different TSN solutions which offer a wide range of services from real-time capabilities to redundancy management for the communication layer of the virtualized integrated system [18, 19, 20].

For deterministic message delivery, TSN uses the global notion of time and a transmission schedule which is called Gate Control List (GCL). The GCL is port-specific and determines at each instant of time which message can be sent [12]. In TSN like in other time-triggered protocols, the transmission schedule of TT communication is computed off-line due to the complex nature of the scheduling problem. Despite GCL advantages, the scheduling problem arising from the GCL synthesis is NP-complete. Therefore, it is hard to come up with scheduling algorithms that apply to different network topologies and at the same time are scalable to large TSN systems. For this reason, the optimization algorithms that are intended for the search space exploration of the valid schedules play a key role in the deployment of TSN. This thesis introduces different heuristic scheduling procedures for TT communication in TSN systems. These scheduling algorithms combine the routing and scheduling constraints and generate static transmission schedules using joint constraints in a single-step [21, 22]. Furthermore, our TSN schedulers satisfy the timing constraints of the mission-critical applications while meeting their reliability requirements [23].

1.4 Thesis Contributions

Highly reliable, scalable and deployable networks with strict temporal constraints are inevitable for future cyber-physical systems. Due to widespread usage and success of Ethernet technologies, the Time Sensitive Networking task group introduces a series of protocol extensions to the IEEE 802.1 Ethernet standard. These standards provide real-time capabilities and performance improvements. Simulation environments are intensively used to investigate the correctness and applicability of new protocol suites. This trend is driven by timely and costly verification and validation (V/V) processes of the implementation of protocols on real hardware.

This thesis presents a simulation framework for the TSN standard, which is developed as an Ethernet-based network for mixed-critically traffic in the Riverbed simulator [24]. The simulation framework provides generic simulation components with stable interfaces. Therefore, it can be effortlessly used to simulate different applications and to verify TSN standards with respect to reliability and timeliness. Moreover, it can be easily extended to integrate additional features of TSN and also future changes introduced by the TSN working group. Furthermore, the presented TSN simulator provides a comprehensive simulation platform for modelling, performance and reliability evaluation of TSN networks in absence of simulation frameworks and actual TSN devices which encompass different TSN services such as remote configuration, clock synchronization and time-aware shaping simultaneously.

The TSN simulator develops simulation models of TSN time-based features including IEEE 802.1Qci [25] and IEEE 802.1Qvb [12].To be more specific, our models implement ingress time-based policing and enhancements for scheduled traffic as an extension of the Ethernet standard. IEEE 802.1Qci defines a policing mechanism that is applied to every stream upon reception and grants the filtering decision based on the local time. On the other hand, IEEE 802.1Qbv introduces a novel scheduling mechanism called Time-Aware Shaper (TAS). TAS is based on the Gate Control List (GCL) concept and specifies at each instant of time which frame can be transmitted. These standards use time as a correctness criterion. Our simulation models provide an opportunity to simulate and evaluate the temporal behavior of TSN networks with high precision. The evaluation of TSN time-aware features is performed by using various network performance metrics like end-to-end delay and jitter [18].

To enforce TAS as well as ingress time-based policing, a TSN-capable device requires to be configured before initiating any message transmission. For this reason, the TSN task group proposed three different configuration models in IEEE 802.1Qcc [26]. These models enable dynamic and remote configuration of TSN-capable devices. The configuration process over TSN systems is carried out using the existing network management protocols (e.g. NETCONF). This thesis develops simulation models for a fully centralized configuration process, which is one of the proposed configuration models in IEEE 802.1Qcc. Therefore, the TSN simulator with a central configuration model enables the modelling of dynamic applications (e.g. train inauguration) and system adaptation (e.g. fault recovery). In the fully centralized approach, first, the configuration model computes the port-specific GCLs based on the system structure and specification of applications within a TSN system. Then, it configures TSN capable devices with the generated transmission schedule information remotely. This work measures the performance of the centralized configuration procedure based on simulation models [27].

Apart from the real-time capabilities, in modern cyber-physical systems, safety is considered as the primary concern. Failures in safety-time critical systems like industrial automation systems may lead to high economic losses as well as dangers

for humans and the environment. Therefore, the Time-Sensitive Networking (TSN) task group not only introduced real-time properties to Standard Ethernet, but also developed a novel fault-tolerance mechanism called Frame Replication and Elimination for Reliability (FRER) [28]. FRER offers highly reliable communication for Time-Triggered (TT) traffic by forwarding messages via redundant routes. This thesis incorporates the FRER functionalities in the simulation models that also implements temporal properties of TSN. Additionally, this work presents a fault injection model to verify the correctness and applicability of the FRER module. Using this model, the different faulty behaviours (including transient and permanent errors) are simulated over the emulated TSN network. Then, the impact of the FRER module on the reliability of TT communication is evaluated in the presence of different faults. The evaluation of TSN fault-tolerance capabilities is carried out by measuring the end-to-end latency and the number of packet loss for every TT stream [19].

The hard real-time systems like industrial control networks have strict temporal requirements. To achieve the real-time capabilities, all devices residing in the mission-critical systems need to be synchronized to a specific reference time. Since in such systems time is used as a correctness criterion, the Time-Sensitive Networking (TSN) task group introduced a fault-tolerant and robust clock synchronization mechanism in the IEEE 802.1AsRev standard. The execution of TSN clock synchronization results in a fully synchronized and scheduled network [17]. This thesis presents a simulation model for time-aware systems which contain different functionalities of TSN clock synchronization. The simulation models of time-aware systems are developed on top of our TSN models that support the time-based features of TSN (i.e. IEEE 802.1Qbv and IEEE 802.1Qci standards), since a TSN-capable model uses its local clock for time-based policing and traffic shaping. Moreover, different TSN synchronization modules including Best Master Clock Algorithm (BMCA), Synchronization process and Peer delay measurement are evaluated using our simulation framework. In addition, the behaviour of the TSN synchronization mechanism is studied in the presence of different faulty behaviours [20].

TSN offers determinism within a fully synchronized network using global TT transmission schedules. The scheduling problem of TSN networks like other time-triggered systems is NP-complete. In addition, the ever-increasing number of network devices, switches and links of many Ethernet-based systems results in numerous possible routes and consequently a tremendous number of schedule possibilities for each TT flow. For this reason, most of existing TT scheduling solutions ignore inter-dependence of routing and scheduling problems to simplify the scheduling process. Hence, they derive the design space of system implementations only from scheduling constraints. This strategy limits the capability of former approaches to compute a global schedule of TT communication for several real-time systems. This thesis presents a heuristic scheduling strategy based on genetic and list scheduling algorithms, since the optimization algorithms for the search space exploration are a crucial element for the deployment of TSN. These solutions combine the routing and scheduling constraints and generate the port-specific GCLs using joint constraints in a single-step. The number of scheduling possibilities within the design space that is derived from joint routing and scheduling constraints increases in comparison to the approaches that only use the fixed routing. Thereby, the schedulability is improved significantly by our solutions. Our schedulers also consider the distribution of real-time applications, multicast patterns and precedence constraints of TT flows in the scheduling process. Moreover, the optimized task binding and resource allocation of our scheduling algorithms lead to significant enhancement of TT transmission efficiency and resource utilization compared to the state-of-art solutions. [21, 22].

In addition to the abstractions mentioned above, the majority of TT schedulers assume that the communication infrastructure is fault-free over time. However, this assumption is unrealistic and in practice, the network experiences different failures during message exchange. Hence, the TT schedules generated by the schedulers with this assumption, are not applicable in case of any failure in the network. Due to stringent temporal and safety requirements of real-time systems, TSN addresses the faulty behaviours using a new spatial redundancy mechanism (i.e. FRER). In this thesis, the TSN schedulers which are designed for fault-free mission-critical systems are expanded considering FRER features. FRER improves the availability and reliability of TT communication by forwarding messages from redundant routes. The main goal of our fault- tolerant TSN scheduler is to satisfy all safety-critical applications deadlines while improving the reliability of the system. For this reason, a new technique to evaluate the reliability of safety-critical systems is introduced. This technique assesses the reliability of a mission-critical system using interactions between real-time tasks in forms of TT messages. Our approach also calculates the reliability of message transmission as a function of the reliability of the network component that engages in message delivery. Consequently, it can be determined which components are more critical in the overall reliability of the system. The system designers can benefit from this reliability analysis for selection of network components and also planning of the network [23].

The main contributions of this thesis can be summarized as follows:

- A simulation framework for TSN time-aware policing and scheduling mechanisms. This framework is developed as an Ethernet-based network for mixed-critically traffic and supports the temporal requirements of real-time systems [18].
- Simulation models comprising different FRER functions. These models provide the fault tolerance capabilities that are essential for data communication of safety-critical applications [19].
- Simulation models containing different TSN clock synchronization procedures such as BMCA and peer delay measurement. These models share the same notion of time by participating in the clock synchronization process. However, the time-aware system models have their local clocks with different drift rates [20].
- Heuristic scheduling strategies based on a genetic algorithm and list scheduling approach for TT communication in time-sensitive networks. These schedulers compute the port-specific GCLs by employing joint routing and scheduling constraints [21, 22].
- Scheduling strategies for fault-tolerant TT communication in time-sensitive networks. These scheduling algorithms aim to improve the system reliability using FRER features while satisfying the deadlines of safety-critical applications [23].
- Simulation models for TSN fully centralized configuration process. The central configuration model calculates the schedules for real-time applications and corresponding streams. Then, it configures TSN-capable devices with the NETCONF protocol. Using this model, the required time interval for remote configuration of devices within a mission-critical system is measured and validated against the maximum permissible configuration duration of TSN-capable devices [27].

1.5 Thesis Overview

This thesis is structured into seven chapters. Chapter 1 provides a brief overview of the virtualized integrated system architecture which is introduced to address new requirements of mixed-criticality systems. Additionally, it details the challenges for the communication infrastructure, which aims to interconnect different components of an embedded system with the virtualized integrated architecture. The second chapter outlines the basic concepts that are used throughout this thesis. For this purpose, chapter 2 starts with a brief overview of distributed real-time systems. Then, it explores different aspects of real-time systems such as communication paradigms, timing and dependability requirements. The third chapter provides a detailed introduction to the Ethernet technologies. This chapter studies different features of the Ethernet standard, including packet switching, quality of service, fault-tolerance and synchronization. Chapter 3 lists the network capabilities required by virtualized integrated system architecture and explains why Ethernet is not suitable for such systems. The last sections of this chapter deliver background knowledge about deterministic Ethernet protocols, which are designed to offer temporal properties and reliability requirements for real-time systems.

The fourth chapter introduces several scheduling strategies which are designed to synthesize GCLs for TSN systems. To achieve that, it first discusses the related work in the context of TT scheduling. After that, the scheduling problem of TT communication in TSN networks is formulated, and the joint routing and scheduling constraints considering TSN scheduling are defined. The further sections provide a detailed description of a Genetic Algorithm (GA) and a Heuristic List Scheduler (HLS) which aim to solve scheduling and routing of TSN systems in a single-step. In the last sections of this chapter, the experimental setup is explained, and simulation results are evaluated.

Chapter 5 describes how the scheduling strategies which are introduced in Chapter 4 can be extended to support reliability requirements of mission-critical systems. To this end, it first discusses the related work on fault-tolerant TT scheduling. Then it formulates the scheduling problem of fault-tolerant TT communication. Further sections propose a novel reliability analysis technique and also describe the fault-tolerant GA and HLS, which are designed to meet timing requirements of mission-critical systems while improving the overall system reliability. This chapter is concluded by an extensive evaluation using experimental results.

The sixth chapter presents the simulation models for temporal properties, clock synchronization, configuration and redundancy management which are introduced in TSN standard. To achieve that, Chapter 6 starts with discussing the existing simulation frameworks which aimed to emulate different time-triggered communication infrastructures. Then it specifies the conceptual models which are essential for deployment of TSN features over real-time systems. To this end, Section describes the centralized configuration model used in TSN simulator precisely. After that, it explains how a standard Ethernet end system and switch model are modified in order to accommodate different TSN features. The last section of this chapter provides a detailed description of experimental setup which is defined based on real world use case. Furthermore, it evaluates the simulation results collected from different test scenarios and validates the applicability and correctness of TSN services for the communication layer of the virtualized integrated system.

Chapter 2

Basic Concepts

This chapter describes the basic concepts that are used throughout this thesis. The first section gives a brief overview of distributed real-time systems that are widely deployed to provide a wide range of services. In the second section, the differences between time-triggered and even-triggered communication are described. The third section explains the role of a global time reference in distributed real-time systems. The last section is dedicated to different aspects of dependability concepts in real-time systems such as dependability threats and failure modes.

2.1 Distributed Real-Time Systems

A real-time system comprises several subsystems such as real-time computer systems, controlled objects and human operators. In a real-time system, the state of the system is changed as a result of the progression of time. In particular, the correctness of a real-time computer system is determined not only based on the computational outputs but also by the time instant at which the results are provided [29].

Real-time systems are divided into two categories: hard real-time systems and soft real-time systems. In hard real-time systems like a brake-by-wire system in a train, violating the timing requirements can lead to disastrous outcomes. However, soft real-time systems such as an entertainment system in a train can continue to operate correctly even under a specific amount of delays [29].

Each real-time system which is also called a cyber-physical system aims to provide different services. To achieve this goal, the real-time computer systems exchange data with controlled objects via actuators and sensors. This data communication is used to exchange digital information with other computer systems and analogue and continuous information with the controlled objects. Moreover, the real-time computer system can be distributed. A distributed computer system comprises several nodes which communicate over a real-time networking infrastructure. The node's service can be defined as a set of messages. The node produces these messages by reacting to inputs, the progression of time and its state. Thereby, the node's service tightly depends on the real-time network services [30]. The structure of distributed real-time systems is shown in Figure 2.1.

2.2 Time-Triggered and Event-Triggered Communication

Over the past decades, mixed-criticality systems which contain safety-critical and non-critical subsystems have drawn more attention. In such systems, the safety-critical subsystems use the time-triggered communication infrastructure for message transmissions. In such subsystems, the transmission of TT messages is periodic and deterministic in terms of delivery delay and jitter. Thereby, for each TT message, the

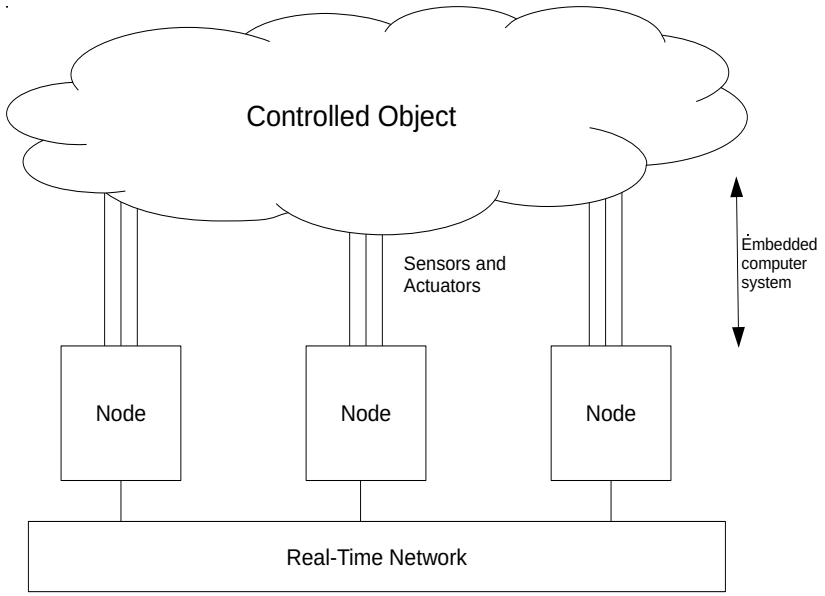


FIGURE 2.1: The structure of distributed real-time system [30]

route from a sender node to the receiver nodes is specified statically at the design time. In addition, for every node on the message forwarding path, the start and end instants of the transmission interval are determined at the design time. Therefore, each node in the time-triggered network uses the predefined schedule for TT message transmissions. The node's static schedule is repeated over least common multiple of cycles of TT messages that are destined to the node. Namely, every TT message is sent at a particular time slot, and this transmission is repeated with the fixed time interval called the message's period [30].

On the other hand, non-critical subsystems exchange messages when no TT message is scheduled. These messages are event-triggered, meaning that the messages are transmitted upon reception of a specific event which is not deterministic. Thereby, for the event-triggered messages unlike TT messages, the start and end instants of the transmission interval are not known. This feature can lead to contention between the time slots of TT messages and event-triggered message transmissions [30]. Figure 2.2 shows an example of a transmission schedule in a mixed-criticality system.

The networking protocols that are used in mixed-criticality systems address the contention arising from the integration of time-triggered and event-triggered communications in different manners. Some protocols like FlexRay [31] reserve a time interval before each TT time slot to avoid interference with even-triggered messages. The length of this time interval is set to the maximum size of an event-triggered message so that the contention avoidance between TT time slots and event-triggered messages is guaranteed. Another group of protocols like TTEthernet, also follow the static contention-free schedule for the transmission of TT messages. However, they do not reserve a guard band before each TT time slot. Instead, if transmission of an even-triggered message interferes with a TT time slot, these protocols preempt the even-triggered message and start transmitting the TT message. After completing the transmission of TT messages, the preempted event-triggered message is resent. The third category of protocols applies neither contention avoidance nor contention detection and preemption. Instead, they are non-preemptive and tolerate contention. This means that if the transmission of an even-triggered message overlaps with a TT

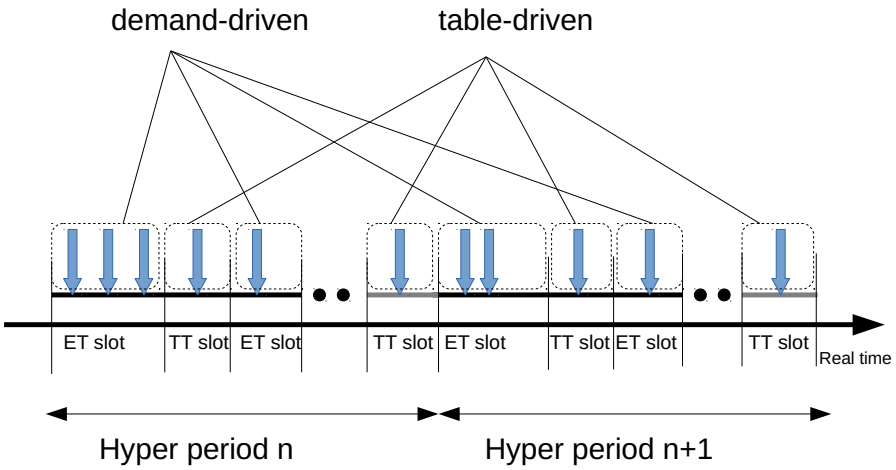


FIGURE 2.2: An example of transmission schedule in a mixed criticality system

time slot, the event-triggered message is not preempted and the transmission continues until completion. Thereby, the transmission of a TT message may be delayed by a maximum of one event-triggered message transmission [30].

2.3 Global Time Base

In distributed real-time systems, time typically described according to Newtonian physics. Thereby, a dense timeline comprises an infinite number of instant, and every event occurs at a specific instant. A node timestamps an event based on its own local notion of the global time. In a dense timeline, it is not feasible to have a fully consistent assignment of timestamps distributed system due to the limited accuracy of the global time and time digitalization. Hence, there is a discrepancy between the timestamps of a certain event on different nodes [32].

To address this issue, the sparse timeline was introduced [29]. The sparse timeline is divided into two time intervals: silence and activity intervals where an event happens during an activity interval. In the sparse timeline, the activity interval presents the granularity for time-triggered activities and the activity duration depends on the precision of the global time. The events which occur during the same activity interval either at the same node or on different nodes are considered as simultaneous. Nevertheless, the events happening in different activity intervals can be ordered consistently based on their timestamps. An example of a sparse timeline is depicted in Figure 2.3.

In distributed real-time systems, time can also be established using an external time reference such as the global positioning system (GPS). The global time tick in GPS is encoded by an eight-byte integer where the five upper bytes are used for the second encoding and the three lower bytes for a fraction of the second [32].

2.3.1 Time and State

Mesarovic et al. [33] defined the state of a system as follow: "The state enables the determination of a future output solely based on the future input and the state a deterministic system is in. In other words, the state enables a "decoupling" of

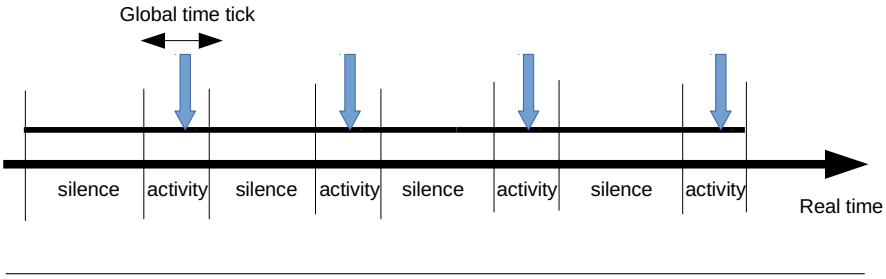


FIGURE 2.3: A sparse timeline [32]

the past from the present and future. The state embodies all history of a system. Knowing the state "supplants" knowledge of the past. For this role to be meaningful, the notion of past and future must be relevant for the system considered."

According to this definition, the state concept highly depends on time modelling. Namely, every state of a system must be represented by a specific global time tick. In distributed real-time systems with the sparse timeline, the silence interval draws a line between the past and the future. This feature enables the distributed systems to mitigate faulty behaviours by replicating the state and voting on the state copies within different fault containment regions [32].

2.4 Dependability

A real-time system is called dependable if it provides the expected services with the desired timing to other systems [30]. For this purpose, dependable services are developed as a series of distributed fault-tolerant functions which operate correctly regardless of faulty behaviours. Thereby, a distributed system that offers real-time and fault-tolerance properties is also called a responsive system [29].

2.4.1 Dependability Threats

Dependability threats can be defined at different levels: When a real-time system does not deliver the expected services, a failure in the system happens. The state of the system which causes the failure is called an error. A fault is associated with a hypothesis that articulates the assumed causes of an error. Depending on the part of a system that causes a failure, the faults are categorized into internal and external faults. The internal faults can be a result of physical faults or design faults either in software or hardware. Similarly, external faults can be caused by physical faults or wrong inputs [30].

2.4.2 Fault-Containment Regions

A fault-tolerant system comprises several subsystems where every subsystem continues to deliver the trusted services even in the presence of faulty behaviours outside of the subsystem. Thereby, each subsystem forms a Fault-Containment Region (FCR) [30]. A fault within an FCR must be detected and mitigated before it impacts the services of other subsystems. For this reason, the error-containment coverage that defines the probability of error detection within an FCR is introduced. The fault-containment regions can be defined in different system levels like hardware units and software components [29]. The separation of FCRs is a critical enabler for building a highly reliable real-time system. Since the redundant units improve the

reliability of the system only if the FCRs are entirely independent, it is noteworthy that shared resources such as power supply and external factors like electromagnetic interference can threaten the separation of the FCRs. In a distributed real-time system, a processing node and a communication link are typically considered as an FCRs where the separation is defined at different levels during the system design [30].

2.4.3 Failure Modes

The impact of a failure in an FCR which are visible to other subsystems in different manners are called failure modes. The system designer must select the appropriate redundancy techniques based on the potential failure modes and the reliability requirements. Some of the possible failure modes in a distributed real-time system can be listed as follows:

- Fail-Stop Failure: occurs when a node stops delivering services. Other subsystems detect this failure and recover it by resetting the node.
- Crash Failure: occurs when a node stops generating outputs. This failure, unlike the fail-stop failure, may stay undetected by other operational nodes.
- Omission Failure: occurs when either a sender fails to dispatch a message or a receiver fails to receive a message. In both cases, a receiver node cannot react to the sent message. The detection of omission failure is not always guaranteed.
- Timing Failure: occurs when a node does not produce outputs during the expected time interval [30].
- Byzantine Failure: As stated in [34], it is the loss of a system service in systems that require consensus". The "two-faced" failure is an example of a byzantine failure.
- Babbling Idiot: occurs when a node does not meet timing constraints. For instance, this failure occurs when a sender transmits messages outside of a predefined time interval.
- Masquerading: occurs when a node sends messages using an other node's identity without its consent [30].

Chapter 3

Ethernet Technologies

For the interconnection of computers, end stations and peripherals residing in a small geographical area like a campus, the Local Area Network (LAN) concept was introduced. In such local networks, network devices join and leave at their will. Ethernet fits well to the LAN requirements such as easy deployment, low cost and high bandwidth. Wide Area Networks (WANs) also use the Ethernet technologies extensively for data communication since they are composed of several LANs [35]. Over the last 40 years, the Ethernet standard evolved to more scalable and flexible solutions by supporting the high demand for bandwidth and different classes of quality. Unlike the majority of uniform networking standards like PROFIBUS [36] and CAN [37] which are developed for specific services and are only applicable to certain systems, Ethernet continues to provide new services and accommodate new requirements for existing and emerging applications [7]. Ethernet was initially defined as a multiple-access network; however, over past years it evolves to an entirely switched network where the networking devices have a point-to-point connection to each other [38]. Ethernet operates in two different modes (i.e. half-duplex and full-duplex), and both modes of operations share the same principles regarding frame formatting and fair arbitration of medium access.

3.1 History of Ethernet Networks

In 1973, the Ethernet network was introduced by Robert Metcalfe, who was an engineer at Xerox. This standard was designed for the experimental setup at Xerox Palo Alto Research Center. The Xerox experimental Ethernet network transmits data at a rate of 2.94 Mbps. After a successful deployment of the Xerox Ethernet setup, the first draft of the Ethernet standard which is also known as IEEE Std 802.3-1985 [39] was proposed jointly by Digital Equipment Corporation (DEC), Intel and Xerox. The IEEE Std 802.3-1985 described the half-duplex Media Access Control (MAC) protocol. Since then, several amendments for IEEE Std 802.3 which aim to support new technologies and different data rates have been introduced. For instance, a full-duplex MAC protocol was developed later in 1997. The fast Ethernet which offered 100 Mbps bandwidth was introduced as the IEEE Std 802.3uTM [40]. Later, the IEEE Std 802.3z [41] and the IEEE Std 802.3ab [42] were dedicated to Gigabit Ethernet and 100 Gigabit Ethernet respectively. It is noteworthy the IEEE Std 802.3-2015 encompasses significant amendments which were defined over the last 40 years [43].

3.2 Packet Switching Approaches in Ethernet Networks

Ethernet switches forward frames using one of the following approaches: In the first method, which is called store-and-forward, the switch first receives the whole frame

and stores it in its memory. Then it calculates the Frame Check Sequence (FCS) of the frame and checks the value with the last field of the frame. The switch forwards the frame to the next hop if the integrity of the frame is preserved according to the FCS check [44].

The second method, which is used by Ethernet switches to forward frames, is called cut-through [45]. In contrast to the store-and-forward switching approach, a cut-through switch starts forwarding the frame immediately after receiving the header and before obtaining the whole frame. Hence, the switch cannot perform the error-check calculation, which requires the whole frame. For this reason, if the frame is corrupted, the invalid frame remains unnoticed and will be forwarded to other parts of the network [46]. To address this issue and also take advantage of cut-through approach, the store-and-forward switches can be used at the edge of the network to eliminate corrupted frames. On the other hand, the cut-through switches can be utilized in the rest of the network in order to speed up the packet switching process. It has to be noted that the integrity of the frame can be deteriorated significantly after passing several cut-through switches which is not acceptable for systems with demanding dependability requirements such as avionic [7].

3.3 Quality of Service Management for Ethernet Networks

Quality of Service (QoS) describes certain quality attributes such as performance or delay. QoS also makes sure that the network satisfies the requirements of every traffic class [4]. The IEEE Std 802.1D [47] amendment was developed to improve QoS within bridged LANs through filtering services. Moreover, the IEEE Std 802.1D enables time-critical communication in bridged LANs by defining expedited traffic profiles. The expedited traffic classes are determined based on the user-defined priority parameters. The filtering services provided by the IEEE Std 802.1D facilitate dynamic configuration of different groups within LANs. Each group is associated with specific segments of a LAN. Thereby, the frames of every group are only sent over the corresponding LAN segments and are filtered by the rest of the LAN segments [48].

The IEEE Std 802.1Q [48] was introduced to advance further the filtering services of IEEE Std 802.1D through a new concept called virtual bridged LAN. In the IEEE Std 802.1Q, a bridged LAN is logically divided into different groups and each group is associated with one or more Virtual LANs (VLANs). This standard simplifies the establishment of virtual subsets of devices and enables adaptation to changes within these logical groups. In addition, the traffic belonging to a particular VLAN is only forwarded over the LAN segment which serves the VLAN [47]. Therefore, network utilization is improved due to the significant reduction in broadcast traffic. This feature allows for more deterministic transmission of time-critical traffic. However, the network resources like bandwidth are still allocated to different applications through statistical multiplexing. Moreover, the frames are transmitted based on a best effort mechanism within a bridged LAN. Hence network congestion is likely to occur. To be more specific, the switches in LAN systems send out frames when the port is idle. Otherwise, they store frames in the egress port queues until the port becomes free again. Thereby, if the queues get full, they start discarding the incoming frames. This behaviour leads to non-deterministic delivery of time-critical messages. Therefore, even a bridged LAN network offers better QoS management, but it cannot satisfy the strict timing constraints of time-critical traffic. To mitigate network congestion, the LAN needs to limit the number of VLANs supported on the links with demanding

usage. In addition, the applications must be allowed only to send a limited number of streams at the predefined cycle. However, imposing such constraints to a LAN is unrealistic since modern cyber-physical systems necessitate several data streams to operate correctly. Furthermore, LANs encounter different faults over time which affect the deterministic behaviour for time-critical communication.

The IEEE Std 802.1Q only provides the logical traffic isolation of different VLANs. Consequently, if the higher priority VLAN frames are sent at, the higher rate, this behaviour results in starvation of the lower priority VLAN traffic. Thereby, the IEEE Std 802.1Q is not equipped with the particular mechanism to address this issue[7].

3.4 Fault-Tolerant Properties of Ethernet Networks

Standard Ethernet is unable to recover from either transient faults such as a corrupted frame due to electromagnetic interference or permanent faults like link failure. Therefore, to address fault-tolerant communication over Ethernet-based networks, a wide range of redundancy management mechanisms are introduced. The IEC 62439 standards are examples of such redundancy mechanisms that were developed for industrial networks with stringent timing and dependability constraints. The IEC 62439 standards are divided into seven parts:

- Part 1: Rapid Spanning Tree Protocol (RSTP) [49]
- Part 2: Media Redundancy Protocol (MRP) [50]
- Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Protocol (HSP) [51]
- Part 4: Cross network Redundancy Protocol (CRP) [52]
- Part 5: Beacon Redundancy Protocol (BRP) [53]
- Part 6: Distributed Redundancy Protocol (DRP) [54]
- Part 7: Ring-based Redundancy Protocol (RRP) [55]

Among the IEC 62439 protocols, only part 3, which describes PRP and HSP, addresses seamless recovery over Ethernet networks. All protocols as mentioned earlier enhance reliability and availability of systems using redundant paths.

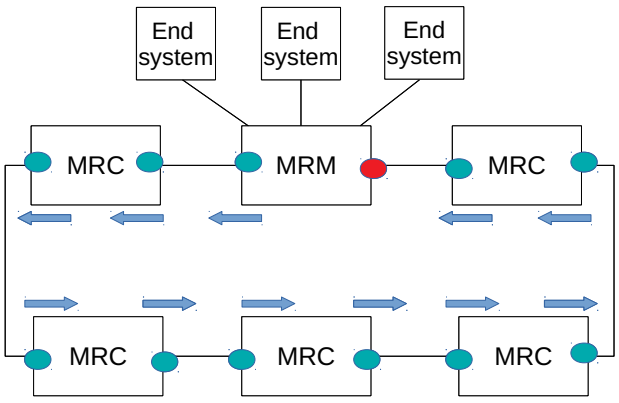
3.4.1 Rapid Spanning Tree Protocol

The Spanning Tree Protocol (STP) enables connectivity between interconnected bridges residing in different LANs. For this purpose, configuration messages are exchanged among bridges. The configuration message carries the spanning tree priority vector, which is used to determine the role of each bridge and also the path cost from each bridge to the root bridge. The role of the root is assigned to the bridge that has the lowest bridge identifier. The bridge identifier is obtained from the bridge priority which is configurable and the bridge address. Therefore, the root bridge has the lowest priority. In addition to the bridge's role, the role of each bridge's port is also specified using the configuration message. The port with the lowest path cost to the root bridge is selected as a root port. On the other hand, the designated port has the lowest path cost from the directly attached LAN to the root bridge.

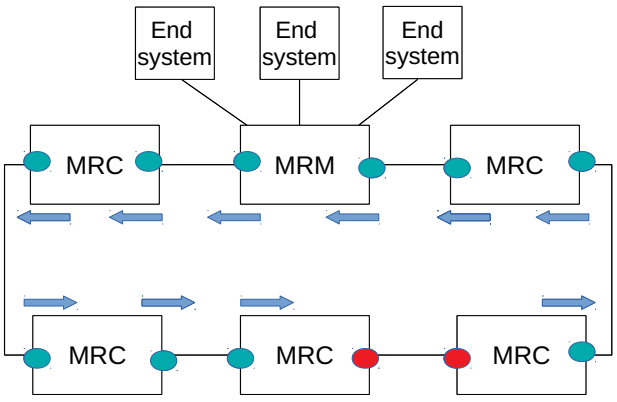
In RSTP, a Single Spanning Tree (SST) which is also called Common Spanning Tree (CST) is established, and the state of each port is identified. Every frame in a

bridged network is switched using the CST regardless of its destination MAC address and VLAN ID (VID). Moreover, RSTP guarantees that there is dependable and symmetric connectivity between every bridge and the directly attached LANs. RSTP also assures that the CST does not contain any loop even in case of a network reconfiguration which is triggered by faulty behaviours and joining or leaving specific devices.

RSTP was designed for the rapid recovery from disconnectivity occurring in a CST. For this purpose, new roles are assigned to each port of a bridge. Then the new root and designated ports start forwarding frames instantly. Namely, a new root port initiates message transmissions irrespectively of received messages from other bridges whereas a new designated port waits for a message from the bridge attached to the LAN to start message switching. Thereby, RSTP finds an alternative path for the existing active route after any failure within up to 2 seconds. Thus, this approach is not applicable for safety-critical systems with stringent temporal and dependability constraints [49].



(A) MRP in a ring-open status



(B) MRP in a ring-closed status

FIGURE 3.1: MRP ring[56]

3.4.2 Media Redundancy Protocol

Media Redundancy Protocol was developed for PROFINET networks based on the Hiper ring concept. In an MRP ring, one node plays the role of a manager (i.e. Media Redundancy Manager (MRM)) whereas other nodes are Media Redundancy Clients (MRCs). In a fault-free network, only one of the MRM's ports forwards the messages while all ports of the MRCs are in the forwarding state. In case of a failure, another port of the MRM is going to the forwarding state and provides connectivity between different ring nodes [56]. It has to be noted the MRM sends test frames to ensure connectivity and integrity of the ring. Thereby, if the test frames are not received by other ports of the MRM, the ring network requires reconfiguration [57]. The reconfiguration of a PROFINET network which supports MRP takes a maximum of 500 ms [56]. Figure 3.1 demonstrates an example of a ring network that supports MRP.

3.4.3 Parallel Redundancy Protocol and High-availability Seamless Protocol

As stated before, part 3 of IEC 62439 introduces two seamless recovery mechanisms based on message duplication and transmission over at least two disjoint paths. Therefore, if any network components such as links and bridges residing in one of the disjoint paths fail, duplicated messages can reach the destination through other disjoint paths. This approach leads to zero recovery time and no frame loss, which is crucial for safety-critical systems [51].

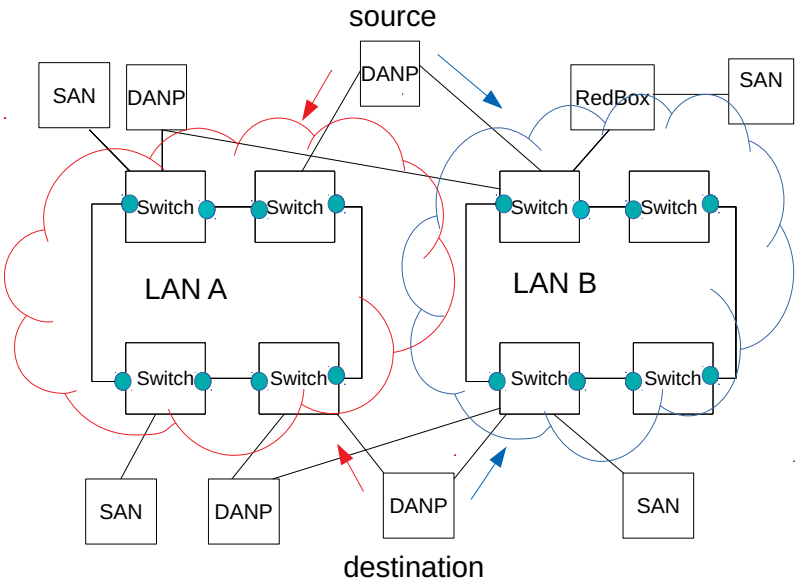


FIGURE 3.2: PRP network [58]

In PRP, nodes have two network interfaces which are connected to two separate networks, as shown in Figure 3.2. Thus, they duplicate messages and send them over two independent networks via separate interfaces. Consequently, if one copy of the frame does not reach the destination node due to a failure (e.g. link failure) in one of the parallel networks, the second copy will be delivered to the receiver node from another network without any interruption. On the contrary, if no failure has occurred within two isolated networks, the destination node only accepts the first copy of the frame and drops the second copy. In PRP-aware networks, a node

attached to two independent networks is called Doubly Attached Node (DAN) while a node is called Single Attached Node (SAN) when it is attached to either one of the networks or two isolated networks via a Redundancy Box (RedBox) [58].

The node with PRP has a layer two link redundancy module. This module duplicates the message received from the higher layers (e.g. application layer) and adds the Redundancy Control Trailer (RTC) to the frame. Lastly, it recomputes the CRC checksum of the message due to the additional trailer (i.e. RTC) and passes the message to the lower layer. The PRP frame format is depicted in Figure 3.3. The PRP redundancy trailer, as shown in Figure 3.3 comprises four parameters:

1. Sequence Number which specifies the order of the messages and will be identical for duplicated frames.
2. LAN A/B that determines the LAN the message is destined to.
3. Link Service Data Unit (LSDU) which indicates the size of a frame including the RCT.
4. PRP suffix that has the value of 0x88FB.

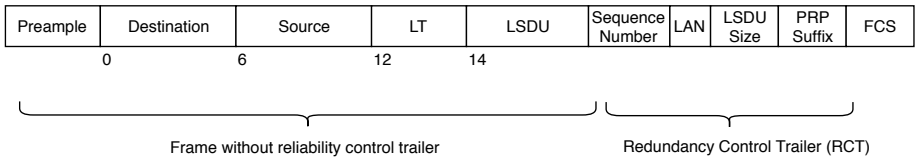


FIGURE 3.3: PRP Frame Format [58]

On the reception side, the link redundancy module keeps track of the sequence numbers of incoming messages. Then it uses this knowledge to discard the duplicated frames [59].

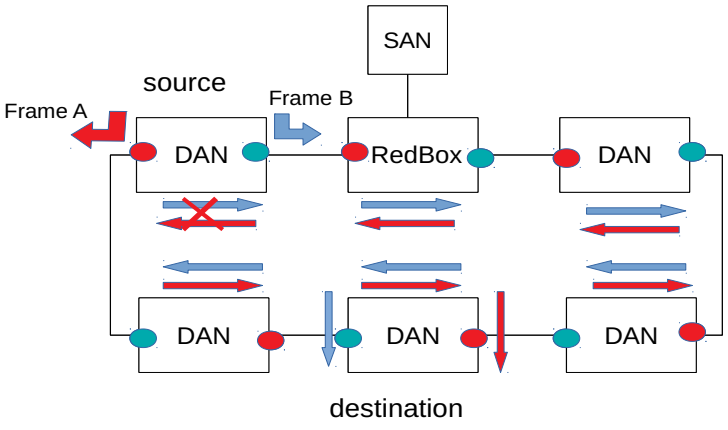


FIGURE 3.4: HSR network [58]

The maintenance of the two networks is cumbersome. For this reason, HSP uses the same approach as PRP, but instead of using two separate networks, it transmits duplicated frames over two redundant paths within one network. An example of an HSR network is depicted in Figure 3.4. HSR was mainly designed for a network with ring topology. Thereby, in an HSR-aware network the sender node dispatches

duplicated frames in two directions while the receiver node accepts the first copy of the frame and filters out the duplicated frame [58].

As it is shown in Figure 3.5, the HSR redundancy header has few differences with the PRP RCT header. For instance, in the HSR header, unlike the PRP header, the HSR ID comes first. It has to be noted that the node without HSR capability cannot be added to an HSR-aware ring network. Since every node in an HSR network needs to send frames via two directions and this is not feasible for SAN. This problem can be resolved by connecting a SAN to a Redbox. In addition, two HSR rings can be interconnected via a device called QuadBox. A QuadBox connects two rings via its four ports [59].

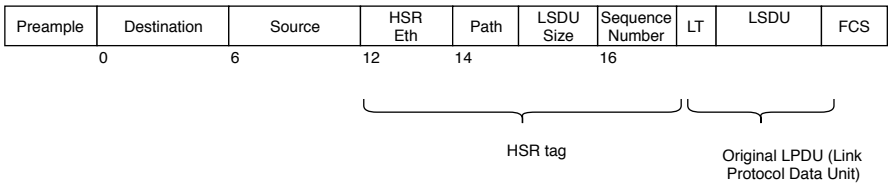


FIGURE 3.5: HSR Frame Format [58]

3.4.4 Cross-Network Redundancy Protocol

Honeywell Corporation introduced the IEC 62439-4 Cross-Network Redundancy Protocol (CRP). CRP is based on fieldbus networks and uses a concept of duplicated networks for redundancy management. In CRP, unlike MRP, the end systems run the redundancy protocol instead of switches, and each end system makes the switchover decision independently. In addition, an end system with CRP can connect to one of the duplicated networks through its own network adapters [52]. An example of a CRP-capable network is illustrated in Figure 3.6. In CRP-aware networks, every frame dispatched from a source node must pass through the inter-LAN links on the path to a destination [57].

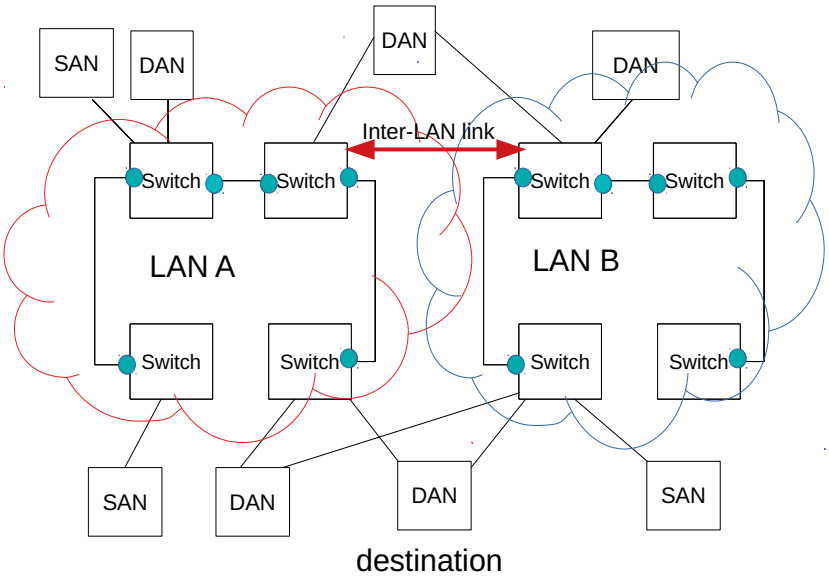


FIGURE 3.6: CRP network [57]

3.4.5 Beacon Redundancy Protocol

The IEC 62439-5 Beacon Redundancy Protocol (BRP) was introduced by Rockwell corporation. However, BRP shares the same redundancy principles as CRP, but it reduces the recovery time using beacon nodes [57].

3.4.6 Distributed Redundancy Protocol

The IEC 62439-6 Distributed Redundancy Protocol (DRP) was designed to offer fault-tolerance properties for real-time systems. For this reason, DRP-aware switches share the same notion of time using IEEE 1588 Precision Time Protocol (PTP) [60]. They are interconnected in a ring topology and follow the same principles as MRP [61].

3.4.7 Ring-based Redundancy Protocol

The IEC 62439-7 Ring-based Redundancy Protocol (RRP) introduced the redundancy management principles for the Real-time Automation Protocol for Industrial Ethernet (RAPIEnet). RAPIEnet was designed for real-time industrial networks with line and ring structures. Moreover, RAPIEnet does not require any external switching functionality. Thereby, such networks can be reconfigured instantly when a device is added or leaves the network. RRP was defined based on a concept called redundancy programmable logic controller (PLC). The PLC contains redundant expansion drivers and a redundant CPU. Each CPU communicates with other devices via RAPIEnet. On the other hand, the expansion base driver supervises devices residing in the expansion base with the help of the CPU [62].

3.5 Clock Synchronization Protocols for Ethernet Networks

In conventional Ethernet-based networks, the time attribute was only used for performance measurements, not for time synchronization. The synchronized global time, however, can be achieved through Global Positioning System (GPS) receivers without any internal synchronization method, but this is not suitable for several real-time systems. Therefore, to transport timing information over Ethernet networks a wide range of protocols such as Network Time Protocol (NTP) [63], IEEE 1588 Precision Time Protocol (PTP) [60], IEEE 802.1AS [64] and IEEE 802.1AS-Rev [17] were developed. Conversely, NTP is not suitable for many industrial applications due to its synchronization accuracy in the order of milliseconds.

3.5.1 IEEE 1588 Standard

The IEEE 1588 Standard, which is also called Precision Time Protocol (PTP) aimed establish a synchronized global clock with sub-microsecond accuracy for measurement and control applications. To be more specific, the IEEE 1588 Standard offers the following services for the measurement and control applications in mission-critical systems:

- Scalable to large mission-critical systems
- Timing accuracy in the range of microsecond to sub-microsecond
- No administrative overhead
- Implementable for devices with different levels of complexity

- Suitable for dependable systems

The first draft of the IEEE 1588 (PTPv1) was published in 2002 and drew attention of different industries such as telecommunication and military. As a result, the second draft of the IEEE 1588 was introduced in 2008 so that it can accommodate the requirements of existing and emerging industrial applications [60].

Table 3.1 presents a comprehensive comparison of state-of-art synchronization mechanisms with the IEEE 1588 standard. As stated in Table 3.1, PTP, unlike IRIG-B, 1PPS and GPS transmits timing information over the data networks such as Ethernet instead of a dedicated network. In addition, the IEEE 1588 protocol, unlike NTP, offers sub-microsecond synchronization accuracy through the peer-to-peer delay measurement mechanism [65].

Mechanism	Typical accuracy	Offer time and date of day	Additional network not needed	Scalable
IRIG-B (AM)	1ms	X		
IRIG-B (DC-shifted)	100µs	X		
1PPS	100µs			
GPS	1µs	X		
NTP	1-10ms	X	X	
PTPv1	1µs	X	X	
PTPv2	1µs	X	X	X

TABLE 3.1: Characteristics of different synchronization methods [65]

Since IEEE 1588v2 is an extension of IEEE 1588v1 and covers all the basic concepts in PTPv1, the following sections focus on the IEEE 1588v2 standard. In addition, to simplify the notion of IEEE 1588v2, in the rest of the report, it is referred to as PTP.

3.5.2 PTP Message Types

PTP is used to synchronize the local clocks of different devices to the global clock through PTP message transmissions. There are two types of PTP messages:

1. **Event Message**: A time-aware system creates a time-stamp for event messages either at egress or ingress units. In short, an accurate time-stamp specifies the point of time an event message is transmitted or received.

2. **General Message**: Unlike event messages, general messages are not time-stamped.

The PTP event messages include:

- **Sync Message**: This message transfers a precise transmission time-stamp which is generated by a grandmaster clock to the slave clocks. Consequently, slave clocks can use this message to calculate the transmission delay between grandmaster and slave clocks.

- **Delay_Req Message**: This message is used to request a receipt time-stamp from the receiving node. A Delay_Req message also transfers an accurate transmission time-stamp which is generated at a slave port.

- **Pdelay_Req Message**: This message is used to measure the link delay between two PTP ports that support the peer delay measurement technique. A Pdelay_Req message transfers an accurate transmission time-stamp which is generated at a PTP port and it is sent to another PTP port.
- **Pdelay_Resp Message**: Upon reception of a Pdelay_Req message, a Pdelay_Resp message is sent to the sending PTP port. The timing information related to a Pdelay_Resp message can be transferred as follows:
    1. A Pdelay_Resp message carries the time difference between the receipt time-stamp of the corresponding Pdelay_Req message and the transmission time-stamp of the Pdelay_Resp message.
    2. A Pdelay_Resp_Follow_Up message carries the time difference between the receipt time-stamp of the corresponding Pdelay_Req message and the transmission time-stamp of the Pdelay_Resp message.
    3. A Pdelay_Resp message carries the receipt time-stamp of the corresponding Pdelay_Req message. On the other hand, a Pdelay_Resp_Follow_Up message transfers the transmission time-stamp of the Pdelay_Resp message.

The following types of PTP general messages can be distinguished:

- **Announce Message**: This message is exchanged between different nodes in order to build the synchronization hierarchy. In more detail, an announce message encompasses the specifications of the sender and the corresponding grandmaster and is used by a receiver to select the best master clock.
- **Follow_Up Message**: This message transfers the transmission time-stamp of the corresponding Sync message in a two-step clock process.
- **Delay_Resp Message**: This message transfers the receipt time-stamp of the corresponding Delay_Req message.
- **Pdelay_Resp_Follow_Up Message**: This message transfers the transmission time-stamp of the corresponding Pdelay_Resp message where a two-step clock process performs peer delay measurements.
- **Management Message**: This message is exchanged between management nodes and clocks in order to retrieve and update PTP data sets stored in the clocks. The management messages, in addition to the system customization, are utilized to set up and manage faulty behaviours in a system.
- **Signalling Message**: This message is exchanged between clocks in order to achieve aligned communication parameters.

In short, the Sync, Delay_Req, Follow_Up and Delay_Resp messages transfer the timing information that is required for clock synchronization and delay measurement. On the other hand, the Pdelay_Req, Pdelay_Resp and Pdelay_Resp_Follow_Up messages carry the timing information that is required for link delay measurements. The ordinary and boundary clocks supporting the peer delay measurement synchronize to the global clock through the timing information in the Sync and Follow_Up messages and the link delay.

3.5.3 PTP Device Types

The PTP devices are:

- Ordinary Clock
- Boundary Clock
- End-to-end Transparent Clock
- Peer-to-Peer Transparent Clock
- Management Node

PTP defines two methods for measurement of the propagation delay:

1. Delay request-response method
2. Peer delay method

The ports of the ordinary and boundary clocks are capable of implementing both delay measurement methods. However, the ports of the peer-to-peer clocks can only support the peer delay method. On the contrary, these methods do not apply to the end-to-end clocks.

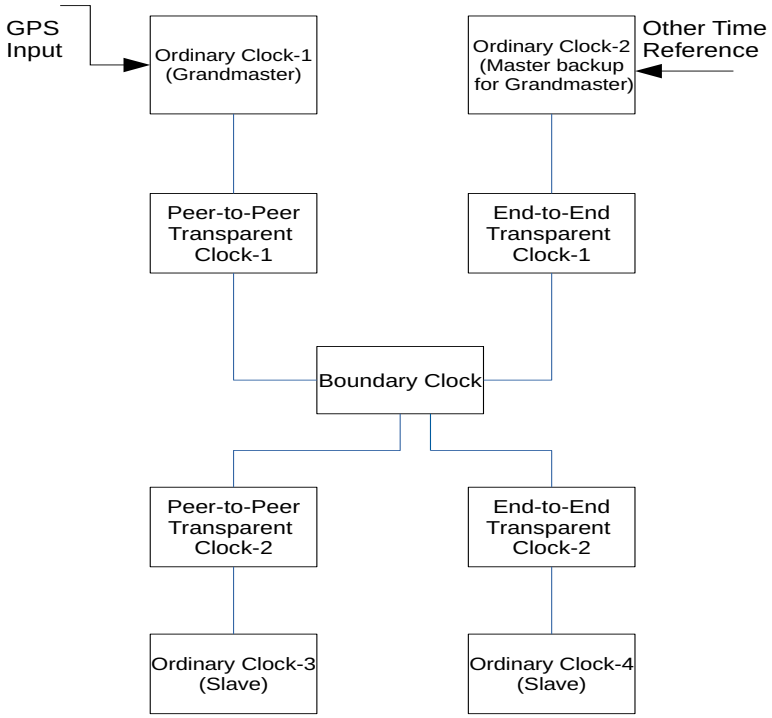


FIGURE 3.7: Network of PTP Clocks [60]

3.5.3.1 Ordinary Clock

In PTP, an end-device may act as an ordinary clock with either role of grandmaster, master or slave clock. Thereby, an ordinary clock can only transmit or receive PTP messages. If the port of an ordinary clock is in the slave state, the local clock is synchronized to the corresponding master clock. In contrast, if the port's state is master

and the ordinary clock plays a grandmaster clock role in the network, the local clock is synchronized to an external source of time (e.g. GPS). Figure 3.7 illustrates the example of the PTP clock hierarchy.

3.5.3.2 Boundary Clock

In PTP, a bridge acts as a boundary clock. As denoted in Figure 3.8, a boundary clock plays the role of master and slave clocks simultaneously depending upon the device that it is connected to. Namely, if a boundary clock is connected to a master clock, it acts as a slave clock whereas it is a slave clock for an attached device with the role of a master clock. The PTP messages associated with the synchronization process are terminated at a boundary clock and recreated with the updated header. However, a boundary clock forwards management and non-PTP messages in a similar way to a standard bridge.

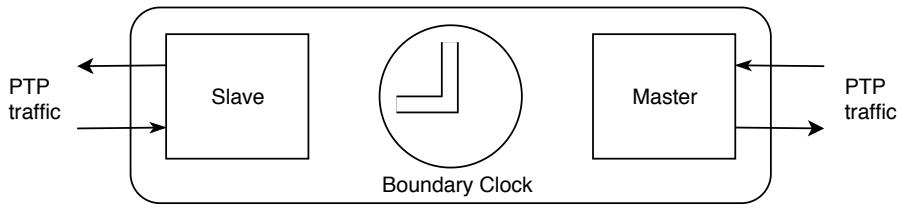


FIGURE 3.8: PTP boundary clocks [60]

3.5.3.3 End-to-End Transparent Clock

To achieve accurate clock synchronization, an end-to-end transparent clock first measures the residence time for PTP event messages that traverse the clock. Then, it updates the correction field of these messages, or follows up messages based on the residence time. The residence time specifies the time interval that event messages remain in a transparent clock. Therefore, a slave clock uses the correction field to calculate the absolute time difference between its local clock and a master clock. For measuring a residence time, a transparent clock time-stampes event messages both on arrival and transmission based on its local clock. However, a transparent clock does not synchronize to a master clock. Consequently, it is quite likely that the measured residence time is inaccurate due to rate differences of a transparent clock and a master clock. The impact of inaccuracy in residence time measurement can be diminished through either synchronization to a master clock or using a free-running local clock. Figure 3.9 depicts how residence time is computed in an end-to-end transparent clock.

It is noteworthy that an end-to-end transparent clock acts as a standard bridge for other messages. This means it forwards messages without further processing.

3.5.3.4 Peer-to-Peer Transparent Clock

A peer-to-peer transparent clock similarly to an end-to-end transparent clock enhances the accuracy of the clock synchronization process through correction of event messages. However, as shown in Figure 3.10, a peer-to-peer transparent clock accumulates the correction field not only based on residence time but also considering link delay. In addition, unlike an end-to-end transparent clock which corrects timing information of all PTP event messages, a peer-to-peer transparent clock modifies only the correction field of Sync and corresponding Follow_up messages.

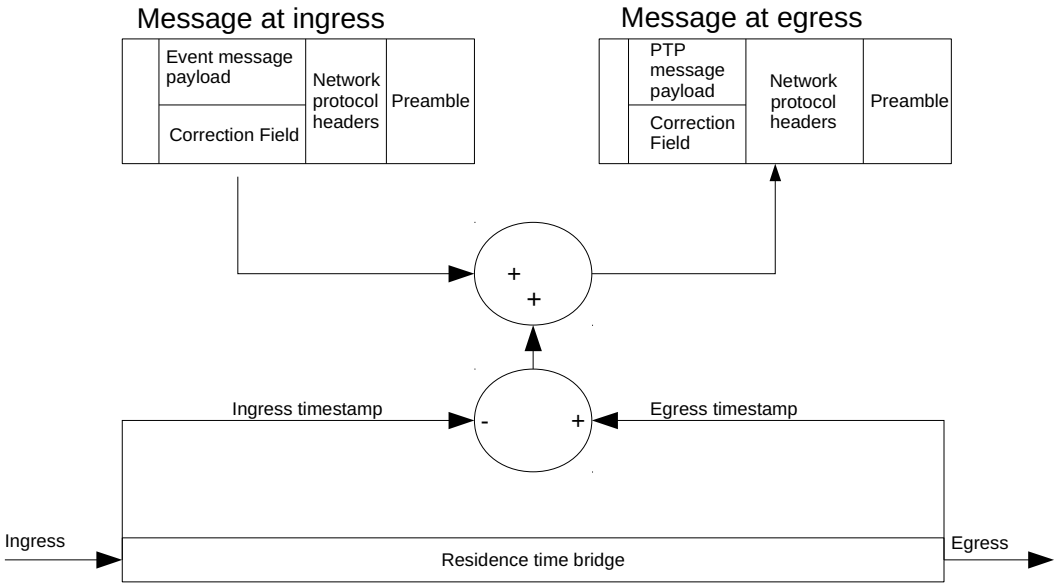


FIGURE 3.9: PTP end-to-end transparent clock [60]

A peer-to-peer transparent clock calculates the propagation delay corresponding to the link which is located between the ingress port and the neighbouring port and traversed by Sync messages. Since a peer-to-peer transparent clock can only attach to the devices which are equipped with the peer delay measurement mechanism, it can only communicate with an end-to-end transparent clock through boundary clocks. Moreover, a master clock running on a network with peer-to-peer transparent clocks dispatches only Sync, Follow_Up and Pdelay_Resp messages which leads to lower network load compared to the network with end-to-end transparent clocks. A Sync message that is forwarded over a network with peer-to-peer clocks carries the residence time and link delay associated with the forwarding path. Thereby, a slave clock computes the clock offset concerning the master clock based on timing information that is encapsulated in Sync or Follow_up messages. On the contrary, in a network with end-to-end transparent clocks, a slave clock apart from Sync messages requires Delay_Resp messages to adjust the local clock. As a result, in the latter case, the clock synchronization procedure takes more time to complete.

3.5.3.5 Management Node

A management node handles management messages through human or programmable interfaces. In addition to the management role, it may also act as any of the clocks mentioned above.

3.5.4 Synchronization in PTP systems

In PTP systems, the synchronization process is performed in two phases:

1. Establishment of the synchronization tree
2. Synchronizing local clocks to a master clock

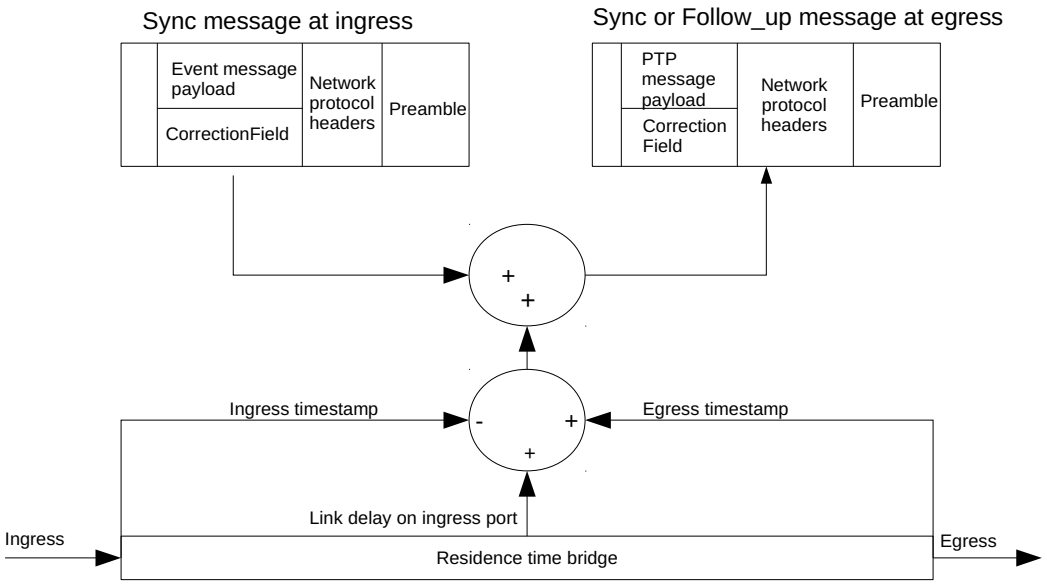


FIGURE 3.10: PTP peer-to-peer transparent clock [60]

In a PTP domain using the best master clock algorithm, first the best source of a clock is selected as a grandmaster clock, and then the state of each port is identified. Each PTP port has one of the following states:

- master: the port provides the reference time for other neighbouring ports.
- Slave: the port synchronizes to a device with a master port.
- Passive: the port acts as neither a master nor a slave port.

After building the master-slave hierarchy, the grandmaster clock transmits Sync messages periodically with correct egress timing information to the slave clocks. In addition, transparent clocks modify Sync messages to provide timing information for end-to-end correction.

3.5.4.1 Best Master Clock Algorithm

In PTP systems, ordinary and boundary clocks send their attributes to all devices through announce messages. Upon reception of Announce messages, a PTP port executes the Best Master Clock Algorithm (BMCA) to specify the clock with better attributes and also the state of the port. For this reason, the BMCA comprises two different algorithms:

- Data set comparison algorithm: Each clock is identified by a data set which contains the following parameters:
  - Priority1 and Priority2: A user configures these parameters through the PTP profile.
  - ClockClass: This parameter determines the traceability of the rate of the master clock.
  - ClockAccuracy: This parameter indicates the accuracy of the clock.

- OffsetScaledLogVariance: This parameter specifies the stability of the clock.
- ClockIdentity: This parameter is used as a unique identifier for the clock.

This algorithm is used to compare the attributes of the local clock with the data sets of other ordinary and boundary clocks which are provided by Announce messages. Furthermore, the data set comparison algorithm compares the data set of the current master clock, which is encapsulated in the grandmaster field of the announce message with the attributes of the local clock. Therefore, the port, after executing the data set comparison algorithm, can identify the clock with better attributes.

- State decision algorithm: Using the outputs of the data set comparison algorithm, this algorithm specifies the state of the port [60].

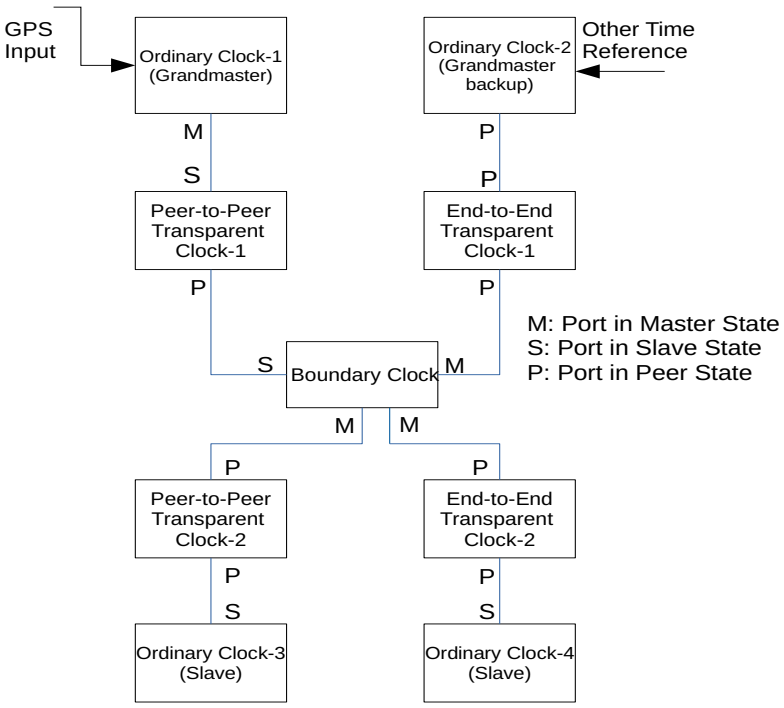


FIGURE 3.11: An example of Master-Slave hierarchy [60]

Figure 3.11 denotes an example of a master-slave hierarchy. In this PTP system, it is assumed OrdinaryClock-1 has an external clock source and possesses the best data set among the clocks. Thereby, its port is in the master state. Likewise, OrdinaryClock-2 is connected to an external clock source. However, its attributes are not better than the ones of OrdinaryClock-1. As a result, it acts as a backup for the grandmaster clock (i.e. OrdinaryClock-1) and its port is in the passive state. BoundaryClock-1 has only one port in the slave state which receives timing information from the grandmaster clock and sends this information to other devices via its master ports. For instance, OrdinaryClock-3, OrdinaryClock-4 and Ordinary Clock-5 adjust their local clocks using the Sync messages provided by BoundaryClock-1. In contrast, OrdinaryClock-2 does not react to the Sync messages from BoundaryClock-1. Since transparent clocks (e.g. peer-to-peer TransparentClock-1) do not execute BMCA, their ports are in the passive state [66].

3.5.4.2 Propagation Delay Measurement

In PTP systems, ordinary and boundary clocks are interconnected through separate communication links. Therefore, slave clocks apart from the Sync messages, requires knowledge about the propagation delay of the forwarding paths to calculate the accurate clock offset with respect to the grandmaster clock. Formula 3.1 denotes how the clock offset is measured:

$$\text{ClockOffset} = \text{Time}_{\text{slave}} - \text{Time}_{\text{master}} - \text{PropagationDelay} \tag{3.1}$$

PTP introduced two methods for propagation delay measurement:

- Delay Request-Response method
- Peer Delay Request-Response method

Delay Request-Response Method This method is designed to compute the propagation delay between two PTP ports, one port in the master state and another one in the slave state. The delay request-response method retrieves the necessary timing information from Sync, Delay_Req, Delay_Resp and Follow_Up messages.

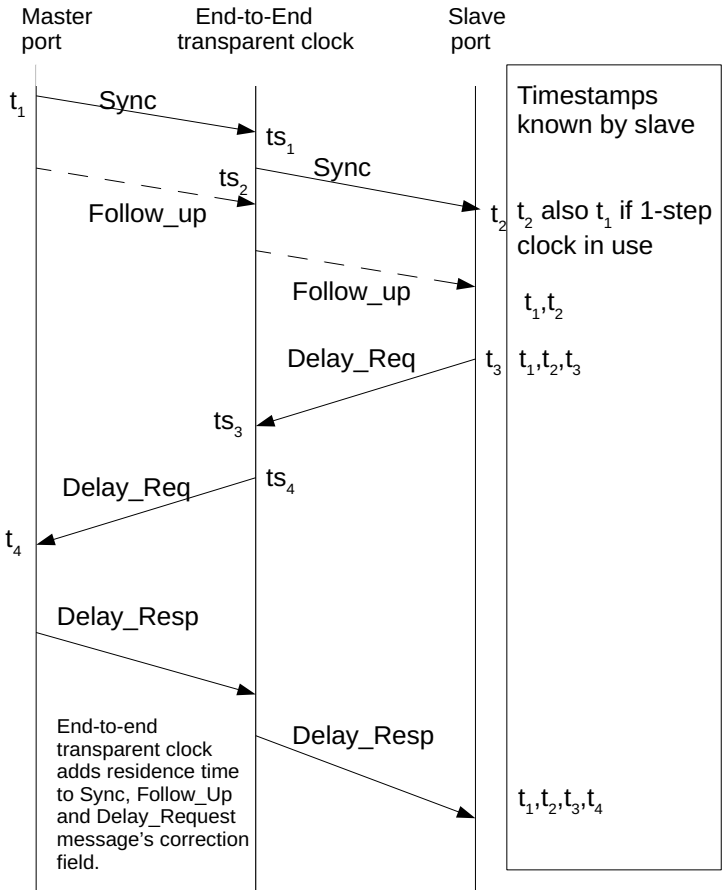


FIGURE 3.12: Delay Request-Response mechanism [60]

Slave clocks synchronize to the grandmaster clock through exchanging PTP messages. As shown in Figure 3.12, the pattern of PTP message exchanges is:

1. The master port transmits a Sync message with the time-stamp t_1 to the slave port. If the master port runs a two-step clock process, the time-stamp t_1 will be sent in the Follow_Up message [60].

2. The Sync message on the way to the slave port goes through the end-to-end transparent clock. This transparent clock generates the time-stamps ts_1 and ts_2 at transmission and receipt unit respectively. Then, it sets the residence field of the Sync message (or the Follow_Up message in case of a two-step clock process) with the value of $ts_2 - ts_1$. If several end-to-end transparent clocks exist between the master port and the slave port, every transparent clock accumulates the residence time in the residence field of the Sync message (or the Follow_Up message in case of a two-step clock process) [66].

3. The slave port upon reception of the Sync message generates the time-stamp t_2. It also retrieves the time-stamp t_1 and the end-to-end correction from the received Sync message (or the Follow_Up message).

4. After reception of the Sync message, the slave port dispatches a Delay_Req message while recording the egress time-stamp t_3 [60].

5. The Delay_Req message on the way to the master port goes through the end-to-end transparent clock. The transparent clock generates the time-stamp ts_3 and ts_4 at transmission and receipt unit respectively. Then, it sets the residence field of the Delay_Req message to the value of $ts_4 - ts_3$ [66].

6. The master port upon reception of the Delay_Req message generates the time-stamp t_4 and sends back the Delay_Resp message with the time-stamp t_4 to the slave port.

7. The slave port upon reception of the Delap_Resp message retrieves the time-stamp t_4 and the end-to-end correction from the Delay_Resp message [60].

After exchanging these PTP messages, the slave port possesses all necessary timing information for computing the mean propagation delay and the clock offset with regard to the master clock as follows:

$$t_1 + t_{offset} + d_{ms} = t_2 \tag{3.2}$$

$$t_3 - t_{offset} + d_{sm} = t_4 \tag{3.3}$$

Where t_{offset} is the clock offset, d_{ms} indicates the mean propagation delay from the master port to the slave port and d_{sm} denotes the mean propagation delay from the slave port to the master port.

As shown in Figure 3.13, the propagation delay is composed of the residence time and the communication channel delay. The residence time determines the time interval a PTP message remains in either a bridge or a switch for packet processing and queuing purposes while the communication channel delay corresponds to the time duration a PTP message traverses the link. Consequently, d_{ms} and d_{sm} can be formulated as follows:

$$d_{ms} = t_{residence_{ms}} + d_{link_{ms}} \tag{3.4}$$

$$d_{sm} = t_{residence_{sm}} + d_{link_{sm}} \tag{3.5}$$

Using formulas 3.4 and 3.5, the equations 3.2 and 3.3 can be written as follows:

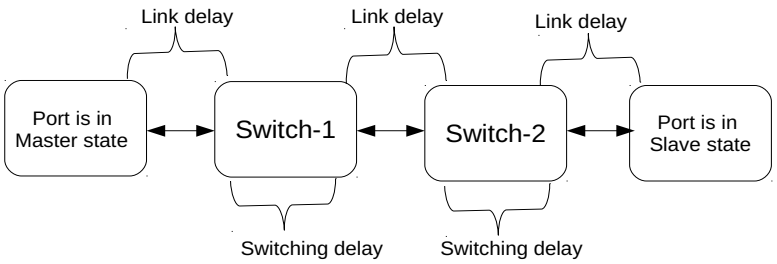


FIGURE 3.13: Propagation delay between master port and slave port [66]

$$t_1 + t_{offset} + t_{residence_{ms}} + d_{link_{ms}} = t_2 \tag{3.6}$$

$$t_3 - t_{offset} + t_{residence_{sm}} + d_{link_{sm}} = t_4 \tag{3.7}$$

If the communication channel delay from the master port to the slave port (i.e. $d_{link_{ms}}$) is identical to the link delay from the slave port to the master port (i.e. $d_{link_{sm}}$), the mean propagation delay can be calculated as follows :

$$d_{MeanPathDelay} = \frac{(t_2 - t_1) + (t_4 - t_3) - t_{residence_{ms}} - t_{residence_{sm}}}{2} \tag{3.8}$$

Using the equations 3.6 and 3.8, the clock offset can be formulated as follows [66]:

$$t_{offset} = \frac{(t_2 - t_1) + (t_3 - t_4) - t_{residence_{ms}} + t_{residence_{sm}}}{2} \tag{3.9}$$

The clock offset computed from the formula 3.9 is used to correct the local clock. The equations 3.8 and 3.9 are written assuming symmetrical forwarding paths between the master port and the slave port. However, it is quite likely that the propagation delay from the master port to the slave port differs from the propagation delay of the reverse direction (i.e. slave-to-master) which leads to an error in the computation of the clock offset [60].

Peer Delay Request-Response Method This method is used to measure the link delay between a pair of PTP ports. Therefore, every two arbitrary ports which implement the peer delay mechanism calculate the link delay regardless of forwarding paths of Sync messages. The value of link delay is later used for end-to-end correction during clock synchronization process [60].

The link delay is measured through exchanging Pdelay_Req, Pdelay_Resp and possibly Pdelay_Resp_Follow_Up messages. As denoted in Figure 3.14, the pattern of PTP message exchange is:

1. Port_1 dispatches a Pdelay_Req message towards port_2 while recording the transmission time-stamp t_1.

2. Port_2 upon reception of the Pdelay_Req message generates a time-stamp t_2.

3. Port_2 sends back the Pdelay_Resp message while generating a transmission time-stamp t_3. Port_2 dispatches the Pdelay_Resp message immediately after reception of the Pdelay_Req message to mitigate the impact of the clock offset

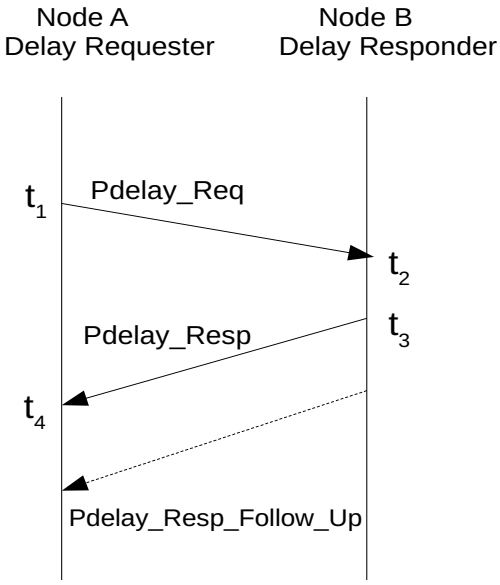


FIGURE 3.14: Peer delay link measurement mechanism [60]

between port_1 and port_2. Port_2 provides the time-stamps t_2 and t_3 to port_1 in one of the following ways:

- the Pdelay_Resp message carries the time difference between the time-stamps t_3 and t_2 (i.e. $t_3 - t_2$).
- the Pdelay_Resp_Follow_Up message carries the time difference between the time-stamps t_3 and t_2 (i.e. $t_3 - t_2$).
- the Pdelay_Resp message and the Pdelay_Resp_Follow_Up message carry the time-stamp t_2 and the time-stamp t_3 respectively.

4. Port_1 upon reception of the Pdelay_Resp message generates a time-stamp t_4.

After exchanging these PTP messages, Port_1 obtains all necessary timing information for the mean link delay measurement. Therefore, it calculates the mean link delay as follows:

$$d_{MeanLinkDelay} = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{3.10}$$

In Equation 3.10, it is assumed that the communication channel between port_1 and port_2 is symmetrical. Thereby, the difference between the propagation delay of port_1-to-port_2 and port_2-to-port_1 results in an inaccurate measurement of the link delay. In addition, if port_1 and port_2 are not synchronized to the grandmaster clock, the time-stamps t_1,t_2, t_3 and t_4 are generated based on different local clocks which leads to an error in the value of the link delay [60].

3.5.4.3 Generation of Message Time-Stamp

Since the local clocks of the slave ports are adjusted to the grandmaster clock using the time-stamps provided by PTP event messages, the accuracy of the clock synchronization highly depends on the time-stamping procedure. In PTP systems, the time-stamping method is implemented either in hardware or software and may be

located at any layer of the Open System Interconnection (OSI) [67]. Namely, hardware time-stamping is typically realized through a dedicated hardware unit at the physical layer, whereas a software time-stamping is implemented at higher layers (e.g. application layer). Therefore, generation of a time-stamp at higher layers is more susceptible to timing errors since a message traverses the lower layers before sending over a network. In contrast, after time-stamping at the physical layer, a message is sent out immediately. Consequently, a hardware time-stamping is more accurate compared to software time-stamping [60].

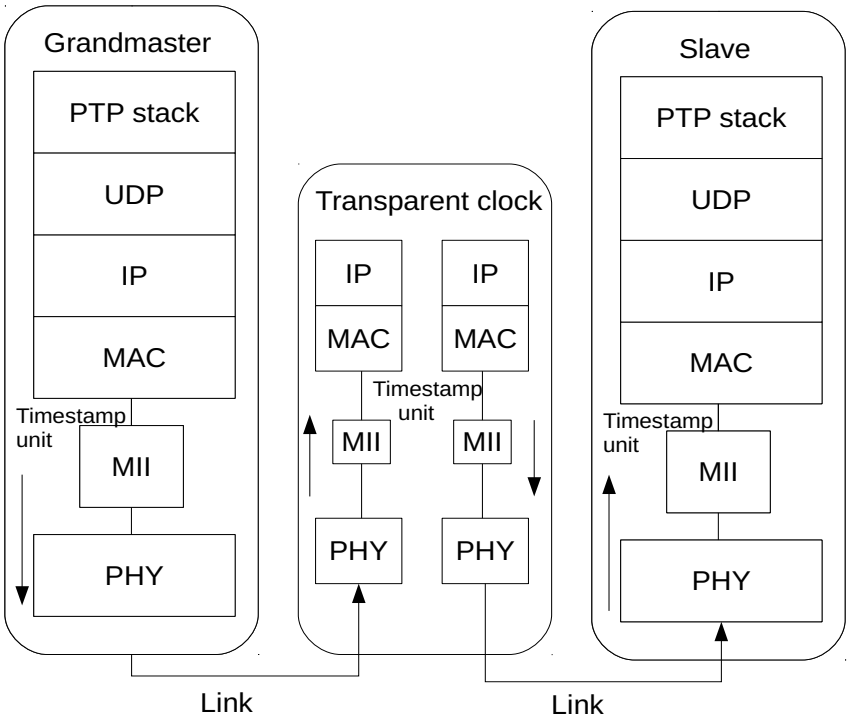


FIGURE 3.15: PTP timestamp generation model [60]

Figure 3.15 presents different PTP devices with associated OSI layers. As illustrated in Figure 3.15, there is a Medium Independent Interface (MII) between the data link layer and the physical layer. The time-stamp unit may be placed at the MII. However, it cannot reside below this interface [66].

3.5.4.4 Transport of PTP Messages in OSI Model

In the OSI model, PTP messages are transported as follows:

- a PTP message over UDP over IP over Ethernet
- a PTP message over Ethernet

In the first case, a UDP datagram carries a PTP message in its payload field. Before sending a PTP message over a network, IP and Ethernet headers are also added to the UDP datagram as denoted in Figure 3.16.

In the latter case, as shown in Figure 3.17, a PTP message comes after an Ethernet header[68].

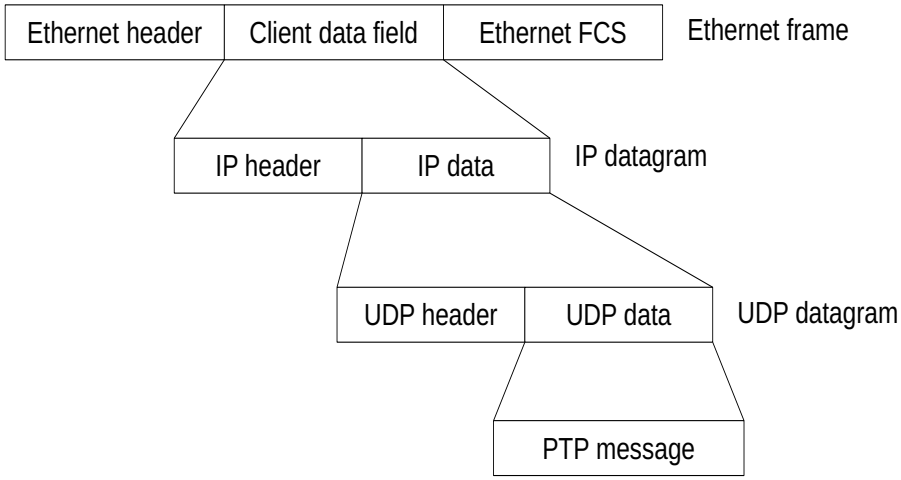


FIGURE 3.16: A PTP message over UDP over IP over Ethernet [68]

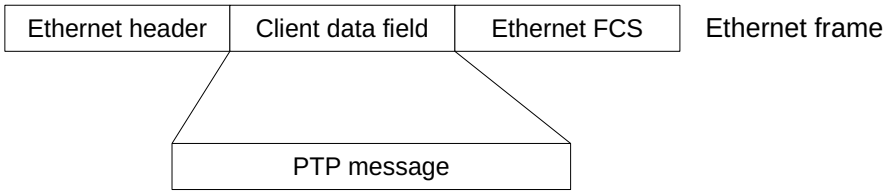


FIGURE 3.17: A PTP message over Ethernet [68]

3.5.5 IEEE 802.1AS Standard

The IEEE 802.1AS Standard [64] is designed to establish and maintain clock synchronization among nodes that communicate over bridged and virtual bridged LANs. In short, the IEEE 802.1AS standard offers synchronization services for time-sensitive applications. Since the IEEE 802.1AS protocol is defined based on the IEEE 1588 standard while adding new specifications, it is also called generalized Precision Time Protocol (gPTP). In other words, this standard enhances the PTP functionalities through new features in order to fulfill the temporal requirements of time-sensitive applications.

3.5.5.1 Time-Aware Bridged Local Area Network

Several nodes attached to gPTP-capable LANs form a time-aware bridged local area network. To be more specific, several time-aware systems attached to LANs that support gPTP services establish a gPTP domain. The time-aware systems are divided into two categories:

- **Time-aware end station**: It is either a grandmaster clock or slave clock.
- **Time-aware bridge**: It receives the timing information of the grandmaster clock, modifies it based on the residence time and the link delay and eventually forwards the modified information to other time-aware systems.

In gPTP systems, there are four types of links and associated delay measurement methods:

- full-duplex point-to-point Ethernet links
- Ethernet Passive Optical Network (EPON) [69]
- IEEE 802.11 wireless [70]
- generic Coordinated Shared Networks (CSNs) [71]

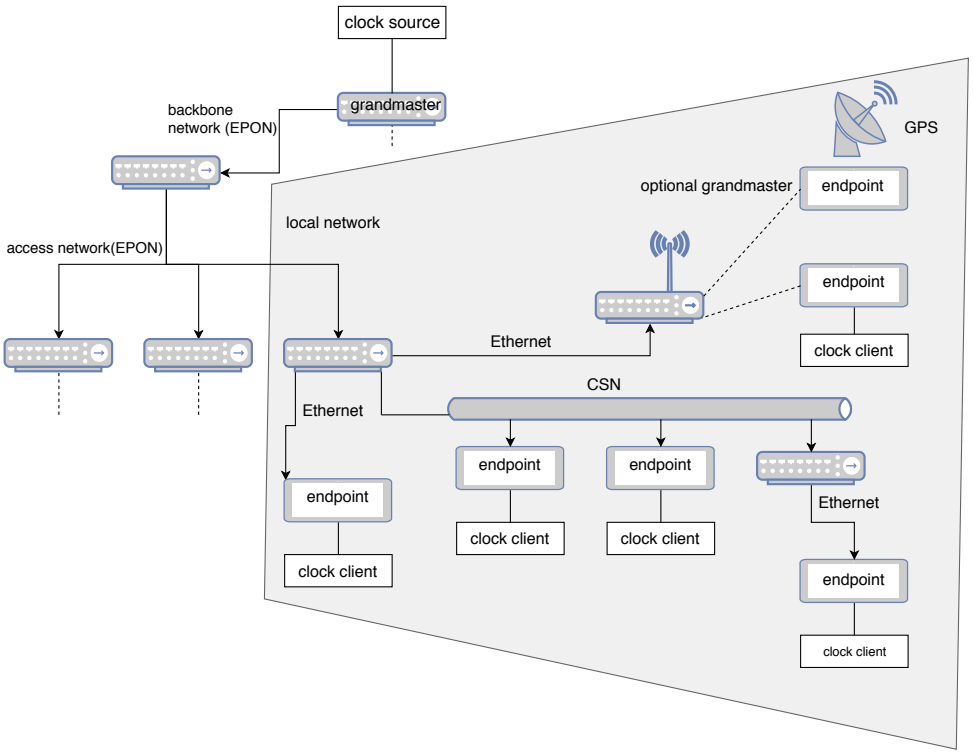


FIGURE 3.18: Time-aware network example [64]

Figure 3.18 depicts a gPTP domain that contains all four types of links described above. In this time-aware network, time-aware systems within the local networks communicate with the grandmaster clock residing in the backbone network through the access network.

In gPTP systems, each time-aware system executes the Best Master Clock Algorithm (BMCA) which is very similar to PTP's BMCA to identify the clock with better attributes. Therefore, after the execution of BMCA, all time-aware systems within a gPTP domain use the same grandmaster clock. For example, in Figure 3.18, the bridge in the backbone network has the best data set, and hence it is selected as a grandmaster clock. However, as shown in Figure 3.19 when the link between the access network and local networks fails, the endpoint connected to the external source of a clock (i.e. GPS) takes over a role of grandmaster and the time-aware network is divided into two separate gPTP domains.

3.5.5.2 Synchronization in gPTP Systems

The clock synchronization in gPTP-capable LANs follows the same principles as in PTP systems. This means a grandmaster clock transmits its timing information to all neighbouring time-aware systems. If the immediate neighbour of a grandmaster is a bridge, it corrects the received timing information based on the residence time and the communication channel delay and then resends it to the neighbouring

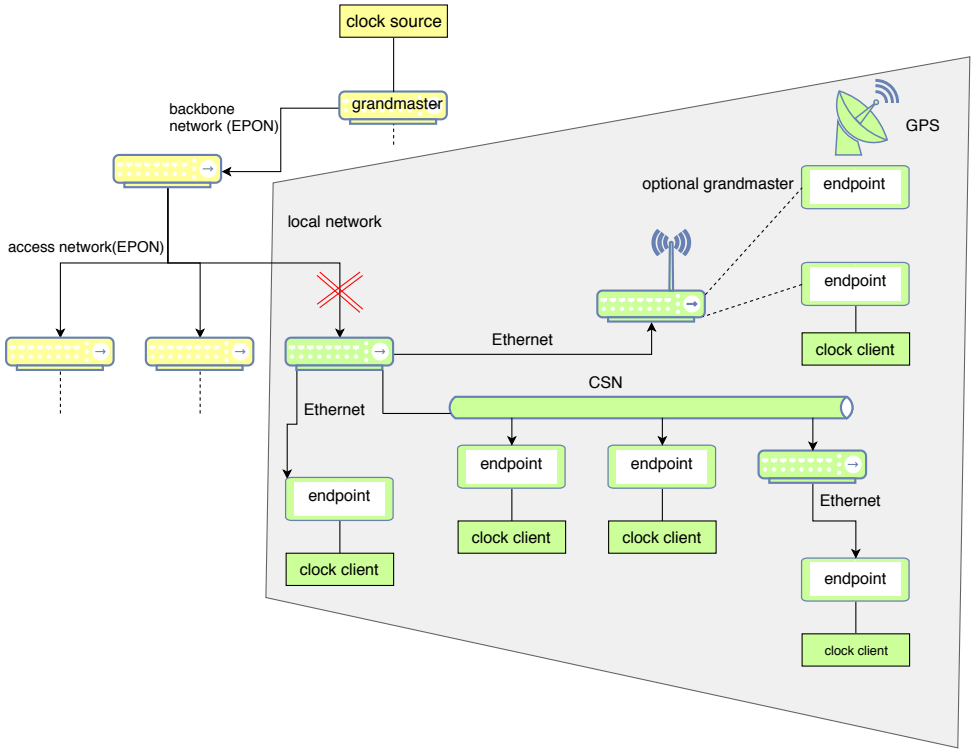


FIGURE 3.19: Time-aware network shown in Figure 3.18 after a link failure [64]

time-aware systems. The residence time is calculated locally in a bridge, while the link delay is measured by exchanging a series of messages between two time-aware systems.

Link Delay Measurement The link delay is measured differently for each type of link that is used to connect time-aware systems in a bridged LAN. However, the basic principles for delay measurement are similar in all network technologies. This means a time-aware system issues a request to a peer system and records the transmission time-stamp. On the other hand, the recipient of the request notes the ingress time-stamp and sends back the receiving time-stamp in a response message. Figure 3.20 illustrates the process of delay measurement.

The delay measurement methods for different network technologies used in time-aware bridged LANs are described as follows:

- For full-duplex Ethernet links, the PTP peer-to-peer delay measurement mechanism is employed.
- For EPON LANs, the discovery procedure is applied by exchanging GATE and REGISTER_REQ messages.
- For IEEE 802.11 wireless LANs, the IEEE P802.11v timing measurement method is utilized.
- A CSN has its own delay measurement procedure, or it uses the PTP delay measurement method.

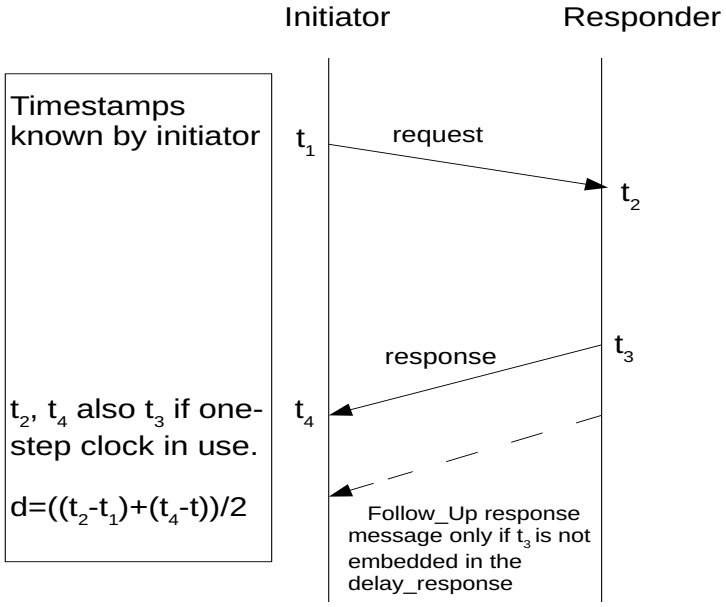


FIGURE 3.20: Simplified delay measurement mechanism [64]

Syntonization The accuracy of clock synchronization depends on the end-to-end correction, which is measured using the residence time and the link delay. Thereby, the egress timing information of time-aware systems is accurate if the local clocks which are used for time-stamping of event messages are frequency locked (syntonized) to the grandmaster. However, syntonization is a timely and error-prone process. Therefore, time-aware systems compensate the clock offsets through the grandmaster frequency ratio.

In gPTP systems, there are two ingredients for adjusting the frequency of local clocks to the grandmaster:

- **Neighbour Rate Ratio (NRR)**: which specifies the frequency ratio of two neighbouring time-aware systems. An NRR is measured for every port of a time-ware system and identifies the frequency ratio of the clock of the neighbouring time-aware system which is directly attached to that port to the frequency of its clock.

- **Cumulative Scaled Rate Offset (CSRO)**: which determines the frequency ratio of the grandmaster clock and a certain local clock. A CSRO is encapsulated in the TLV (Type Length Value) field of a Follow_Up message.

An NRR is used to compensate the rate differences in the link delay measurement method while a CSRO is utilized to correct the clock offset to the grandmaster.

As illustrated in Figure 3.21, the CSRO is calculated from the accumulation of NRRs. There are two major reasons behind the CSRO measurement mechanism:

1. Since NRRs are continuously measured through the peer-to-peer delay measurement method, if a network encounters any changes (e.g. reconfiguration or switching to a new grandmaster), there is no need to compute NRRs again. Besides, upon reception of a first Follow_Up message, a time-aware system retrieves a new CSRO. Consequently, the bridged LANs remain in a transient state resulting from network changes for a shorter time.

2. If a time-aware system introduces an error in the clock offset measurement, it results in an inaccurate measurement of residence time. However, the error

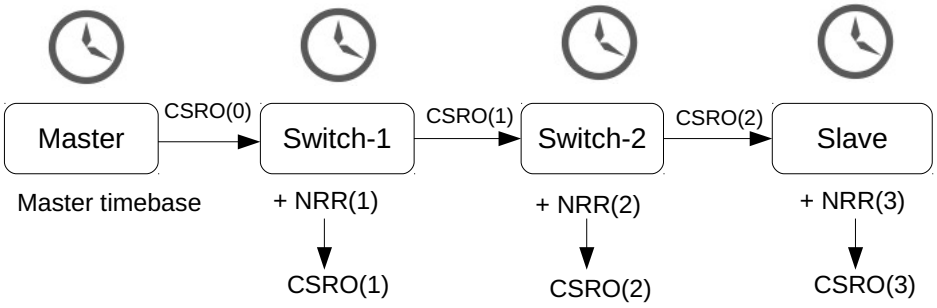


FIGURE 3.21: Relation between NRR and CSRO [64]

in the clock offset value does not have a direct impact on the clock offset of downstream systems.

Grandmaster Selection and Network Establishment For establishing a synchronization tree, all time-aware systems within a gPTP domain execute the BMCA. Since the forwarding spanning tree which is defined by Rapid Spanning Tree Protocol (RSTP) [49] does not serve all synchronization specifications, it typically differs from the synchronization tree.

In gPTP systems to employ synchronization services, all stations and bridges must be time-aware systems since the standard bridges cannot transport the timing information. Each time-aware system examines whether an adjacent node is a time-aware system or not using the peer delay method. For this reason, a time-aware system issues a Pdelay_Req message to the neighbour node and then waits for a response. The time-aware system identifies the neighbour node as a standard bridge if it receives either no reply, multiple replies or the measured link delay embedded in the Pdelay_Resp message exceeding the predefined upper-bound. The time-aware system, however, identifies the peer node as a standard bridge, it continues sending Pdelay_Req messages to the adjacent node periodically in order to identify the peer's gPTP capabilities.

3.5.5.3 Time-Aware System Model

As shown in Figure 3.22, a time-aware system is composed of:

- a time-aware application layer which is either source or recipient of timing information.
- a media-independent layer which comprises ClockMaster, ClockSlave, LocalClock, logical SiteSync and at least one PortSync module. The SiteSync module sends timing information to the logical ports, the ClockSlave and the ClockMaster during BMCA execution. On the other hand, the PortSync modules calculate the communication channel delay.
- a media dependent port maps the abstract MDSyncSend and MDSyncReceive structures coming from the media-independent layer to the appropriate methods for the network technology that the port is directly attached to.

For full-duplex Ethernet ports, the PTP Sync and Follow_Up messages are exchanged for clock synchronization. Additionally, the propagation delay is calculated based on the PTP peer-to-peer delay measurement method.

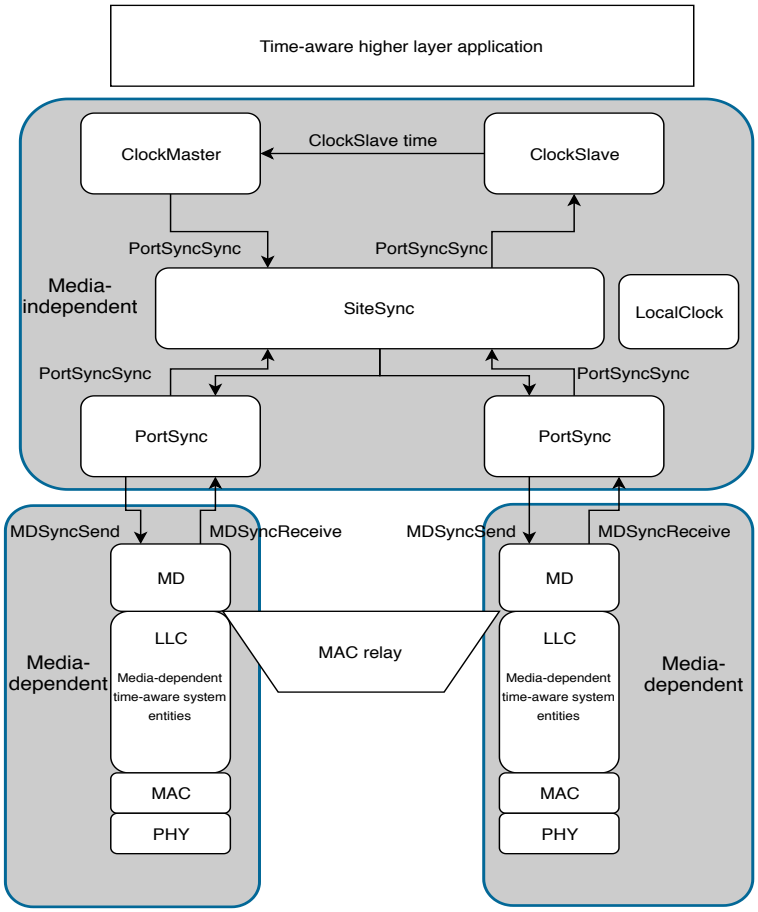


FIGURE 3.22: Time-aware system model [64]

For IEEE 802.11 ports, the MAC module issues a request for timing measurements. The response to a timing measurement request provides all the necessary information required for end-to-end correction.

In EPON LANs, "slow protocol" is used to transport timing information. On the other hand, the timing information is transmitted over CSN through the same communication infrastructure as the Ethernet LANs.

3.5.5.4 gPTP and PTP differences

The key procedures of gPTP share the same principles as the IEEE 1588 standard, but there are several differences between gPTP and PTP:

1. gPTP only supports IEEE 802.3 as the link layer. In contrast, PTP allows different technologies as the link-layer and also higher layers.

2. gPTP facilitates the interconnection of different networking technologies within a single gPTP domain through a media-independent layer. For this reason, the timing information transported across the bridges LANs is generalized, meaning that it conforms to different message structures and management strategies. On the contrary, PTP is limited only to a number of industrial automation control standards such as Ethernet LANs, IPv4 [72] and IPv6 [73].

3. gPTP-capable devices are either a time-aware end station or a time-aware bridge. On the other hand, PTP systems consist of ordinary clocks, boundary

clocks and transparent clocks. A time-aware end station acts as a PTP ordinary clock while a time-aware bridge may act as either a boundary clock or a peer-to-peer transparent clock. Therefore, a time-aware system is capable of measuring residence time and propagation delay and forwarding timing information to other time-aware systems.

4. A gPTP domain comprises time-aware systems since a non-gPTP node cannot participate in the synchronization process. In contrast, in PTP systems non-PTP nodes can forward PTP messages which leads to an additional jitter in end-to-end correction measurement.

5. In Ethernet LANs, time-aware systems employ only the peer-to-peer delay method. However, PTP-aware nodes support the end-to-end delay measurement as well as the peer-to-peer delay method.

6. In Ethernet LANs, a time-aware system supports only the two-step clock synchronization whereas a PTP-aware node supports one-step clock synchronization as well as two-step clock synchronization.

7. A time-aware bridged LAN has only one active grandmaster that leads to a single gPTP domain. On the other hand, a PTP system may consist of more than one PTP domain.

8. In gPTP systems, the syntonization process is compulsory for time-aware systems while this process is not mandatory across PTP-aware networks. Moreover, in PTP, syntonization is a timely process.

9. gPTP's BMCA is very similar to PTP's BMCA in many aspects, but there are some differences between them as follows: (a) A slave port instantly retrieves clock information from Announce messages received from other time-aware systems. (b) A port that is identified as a master port by BMCA goes to the master state instantly. (c) There is no need for the uncalibrated state. (d) All time-aware systems engage in the grandmaster selection regardless of their gPTP capabilities.

10. gPTP provides precise specifications of interfaces that are used in the application layer while PTP does not detail the exchange of timing information between the application layer and other layers.

3.5.6 IEEE 802.1AS-Rev Standard

The IEEE 802.1As-Rev [17] standard apart from the gPTP basic functionalities presents some modification to the IEEE 802.1AS standard. These modifications have been done to accommodate new requirements of modern cyber-physical systems. The major modification is associated with the number of gPTP domains that can be supported by a single time-aware network. The IEEE 802.1AS-Rev standard allows multiple gPTP domains within a time-aware network while the IEEE 802.1AS protocol only supports a single gPTP domain in a time-aware network.

3.5.6.1 Time-Aware Networks with Multiple gPTP Domains

Figure 3.23 depicts a time-aware network with two gPTP domains: domain 0 is the universal time domain, and domain 1 is the working clock domain. All time-aware systems within a gPTP domain which are interconnected via physical links must

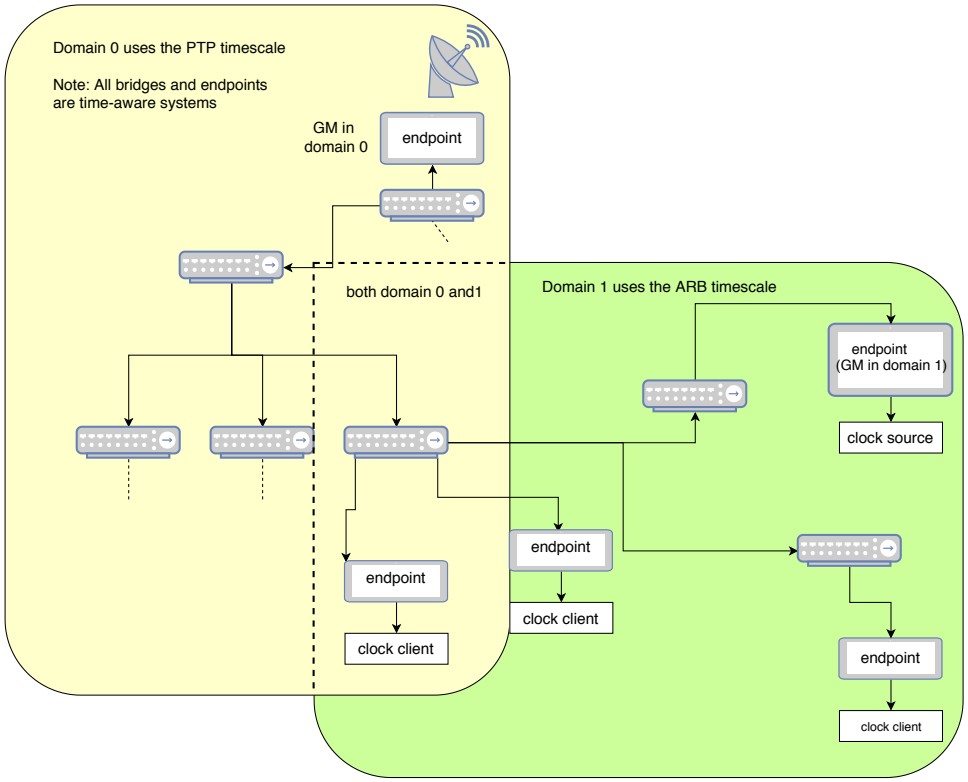


FIGURE 3.23: Time-aware network with multiple gPTP domains [17]. Domain 0 uses a Global Navigation Satellite System (GNSS) as an external clock source, while domain 1 has ARBitrary (ARB) time scale.

support the same time domain. For instance, a time-aware system within the domain 0 cannot connect to other time-aware systems in domain 0 via the time-aware systems that do not support domain 0. Furthermore, a time-aware system which belongs to multiple gPTP domains has a separate gPTP instance for each time domain. Thereby, each gPTP domain executes the BMCA independently and has its grandmaster clock.

3.5.6.2 Time-Aware Networks with Fault-Tolerant Structure

IEEE 802.1AS-Rev is designed for time-sensitive applications in which reliability requirements are essential. To achieve reliable clock synchronization, this standard introduces fault-tolerant synchronization trees which are used to forward timing information. Figure 3.24 illustrates a time-aware network with a grandmaster which is part of two redundant synchronization trees. In this network, the timing information is sent over both synchronization trees simultaneously. Therefore, in case of failure in one of these trees, endpoints and bridges continue to receive the timing data from the redundant synchronization tree.

Aside from the redundant synchronization trees, a time-aware network may have two redundant grandmasters, where one acts as an active grandmaster, and another one is a backup grandmaster. There are two operation modes for a backup grandmaster: hot-standby and cold-standby. A backup grandmaster that operates in the cold-standby mode does not dispatch Sync messages. Nevertheless, a backup grandmaster synchronizes to the active clock regardless of the operation mode since

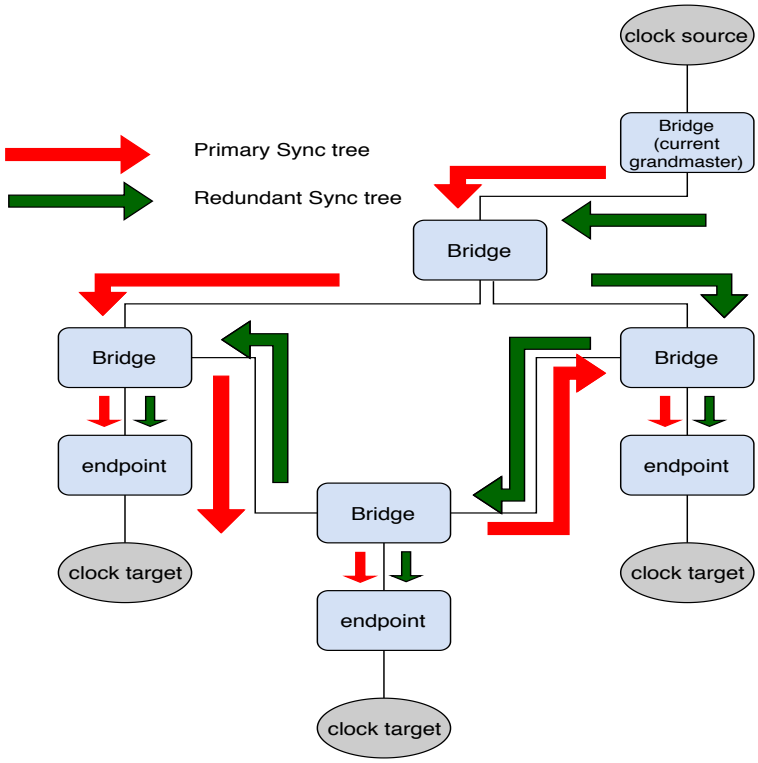


FIGURE 3.24: Time-aware network with redundant synchronization [17]

it is always part of the active clock's synchronization tree. In this standard, it is assumed that the active and the backup grandmasters possess similar synchronization capabilities [17].

3.6 Network Capabilities and Limitation of Virtualized Integrated Systems

As described in Chapter 1, the virtualized integrated system aims to host a wide range of modules on shared embedded resources. Besides, a system with a virtualized integrated architecture can be configured so that every module can communicate with other modules via the network. The data communication can be either synchronous (e.g. control messages), asynchronous (e.g. I/O signals), or best-effort (e.g. video/audio messages). Furthermore, each module in such systems can have different temporal requirements. For instance, some modules can be hard real-time, while others require soft real-time capabilities. The virtualized integrated architecture can be adapted to accommodate different industrial use cases. Any changes in the virtualized integrated system do not have an impact on other existing modules, especially critical ones. This feature enables a cost-efficient V&V process, incremental and modular homologation [7].

3.6.1 Required Network Capabilities of Virtualized Integrated Systems

The communication layer of the virtualized integrated system must provide the following capabilities:

- heterogeneous data communication models so that modules with different requirements can exchange messages over this networking infrastructure.
- robust and fault-tolerant synchronization mechanism that is used for reliable time information distribution among different modules and eventually synchronization of time-critical modules.
- fine-grained QoS management so that each traffic type achieves its timing requirements (e.g. certain end-to-end delay and jitter).
- different redundancy management mechanisms to ensure reliable and robust data communication for safety-critical modules.
- network congestion prevention mechanisms to diminish message loss and guarantee deterministic behaviours [7].

3.6.2 Limitations of Standard Ethernet for Virtualized Integrated Systems

Standard Ethernet with the help of IEEE Std 802.1Q introduced the VLAN concept and different priority classes, but it is still unable to fulfill the requirements of the communication layer of the virtualized integrated system as listed above. However, Ethernet is used for certain domain-specific use cases through traffic profiling and deployment of new mechanisms (e.g. credit-based shaping). Such solutions are only applicable to a limited number of mission-critical systems. Furthermore, any new system's requirements can lead to an additional modification and eventually, costly maintenance and V&V activities.

It has to be noted that the non-functional requirements of the virtualized integrated system such as dependability, reconfigurability and scalability can not be fulfilled if one of the required network capabilities that are mentioned above is not provided. For instance, the absence of temporal isolation prevents hosting the control module, sensors and actuators on the same computational resource. This design decision limits the scalability and reconfigurability of the overall system, which is undesirable for the virtualized integrated architecture [7].

3.7 Deterministic Ethernet Protocols for Real-time Systems

A wide range of standards with real-time capabilities was introduced for Ethernet networks. Each of the deterministic standards is tailored to a particular industry-specific use case as described in Figure 3.25. The different services of the deterministic standards are developed at layer two of the OSI model since the fine-grained control of data streams is only feasible at lower layers like link and physical layers.

It is clear that the deterministic Ethernet-based technologies such as Profinet RT [74] and ARINC [75] offer real-time capabilities that are essential for mission-critical systems, however, the majority of them are not scalable to modern cyber-physical systems with a large number of safety-critical modules [7].

3.7.1 ARINC664

The ARINC664 standard describes the Ethernet-based network of avionic systems which is also called Avionics Full Duplex Switched Ethernet (AFDX). AFDX provides a reliable and deterministic communication infrastructure, which is crucial for the mission-critical functions of the avionics systems. However, in AFDX network components including switches and end-systems are not synchronized to a global

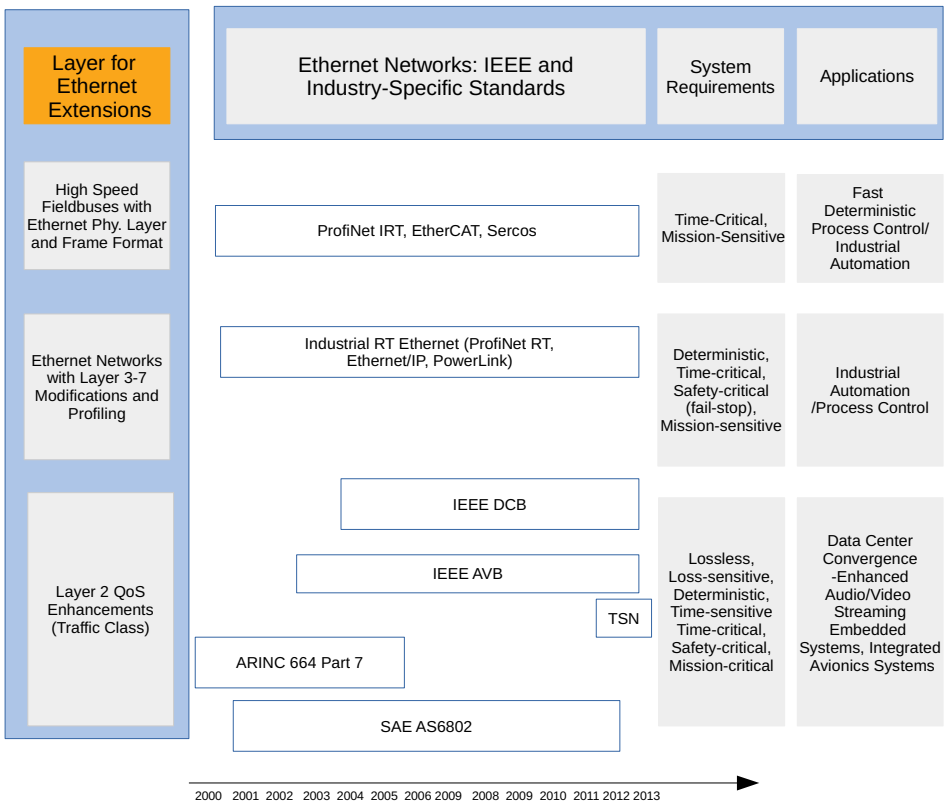


FIGURE 3.25: Deterministic Ethernet variants and protocols for different system requirements and application domains [7]

clock, but still, the transmission and reception of frames follow specific patterns. This feature allows system designers to calculate a maximum latency for every message delivery. Moreover, for scalability purposes, AFDX networks have a cascaded star structure.

3.7.1.1 QoS in AFDX Networks

In avionic networks, every message irrespective of its type must be delivered within a guaranteed delay. Consequently, there is only one class of traffic in an AFDX network which is called the guaranteed service and characterized with the bounded end-to-end latency. To achieve the bounded message delivery delay, guaranteeing a specific bandwidth of links that form the forwarding routes is inevitable.

3.7.1.2 Virtual Link

In an AFDX network, a logical unidirectional connection between a talker and one or more listeners is called Virtual Link (VL). Every VL is identified by Bandwidth Allocation Gap (BAG) and jitter. The BAG specifies the minimum time duration that is required to send two frames from a particular VL consecutively. This implies that in an AFDX network, the traffic shaper only permits the transmission of one frame during every BAG interval. Figure 3.26 depicts the BAG and jitter of an example Virtual Link.

In each frame, the destination MAC address represents the Virtual Link. The Virtual Link header comprises the constant field and the Virtual Link Identifier is

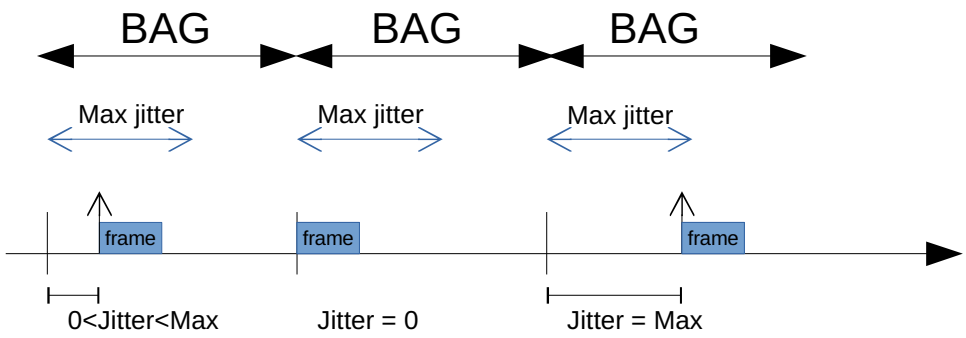


FIGURE 3.26: Jitter Effect for an example data flow [75]

illustrated in Figure 3.27. All end-systems which reside in the same AFDX network must have the same constant field.

48 bits	
Constant field 32 bits	Virtual link identifier 16 bits
xxxx xx11 xxxx xxxx xxxx xxxx xxxx xxxx	

FIGURE 3.27: Virtual Link Identifier format [75]

3.7.1.3 Redundancy Concept in AFDX Networks

In avionic networks, end-systems transmit frames through two or more redundant and disjoint networks. Therefore, the messages are delivered to the receiving end-systems regardless of any failure occurring in one of the redundant networks.

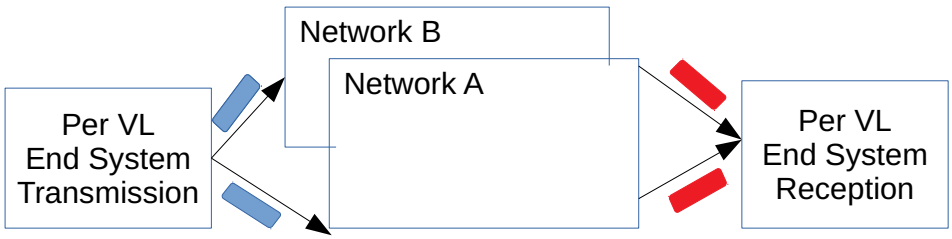


FIGURE 3.28: Network redundancy mechanism in AFDX [75]

In an AFDX network, the redundancy concept is defined for each Virtual Link as denoted in Figure 3.28. This means a sender first adds a sequence number field to every frame and then sends out the frame over multiple independent networks. The sequence number of a specific VL is incremented for each successive frame. On the reception side, an end-system examines the sequence number field of the frame in order to ensure that the frames are delivered in the right order. In addition, a receiving end-system accepts the first frame with the expected sequence number and discards the successive replicated frames.

3.7.1.4 Packet Switching over AFDX networks

Upon reception of a frame, each switch in an AFDX network first enforces different filtering and policing rules regarding traffic rate, frame size, integrity and valid destination addresses. The switch also checks whether the incoming frame carries a valid sequence number and destination MAC address (i.e. VL field). After applying filtering and policing principles, the switch sends out the frame through the proper egress port [75].

3.7.2 Time-Triggered Ethernet

In standard Ethernet, an end-system can transmit frames over the network at any given time, but the frames are only served based on the best effort principles. This event-triggered message delivery leads to non-deterministic end-to-end delay and jitter. Therefore, SAE AS6802 Time-Triggered Ethernet (TTEthernet) [76] is introduced to provide real-time capabilities that are key enablers for mission-critical systems over Ethernet networks. In short, TTEthernet provides a highly reliable and deterministic communication infrastructure for safety-critical applications in the context of automotive, aerospace and industrial control systems. TTEthernet achieves this goal by supporting the time-triggered message transfer where frames are sent at pre-determined instances of time. Static transmission schedule tables are used to support contention-free time-triggered transfers. These tables are generally computed offline using appropriate scheduling strategies.

The time-triggered transfer necessitates a global reference time in order to offer time-triggered services such as temporal isolation and fault detection. Hence, TTEthernet proposes a fault-tolerant and robust synchronization strategy for building and sharing the global notion of time among different network components, including end-systems and switches. The TTEthernet synchronization method at the network initialization phase synchronizes the local clocks of network components to the global clock. It also guarantees the synchronization of the local clocks during the network's operational mode. For this reason, the synchronization strategy reestablishes the synchronized global time in case of loss of the synchronization among the network components. In addition, the synchronization algorithm achieves a high-precision global time by compensating the transmission delay of the synchronization messages [76].

3.7.2.1 TTEthernet Services

The TTEthernet network supports the following services:

Integrate traffic with different level of time criticality. The primary benefit of TTEthernet is the feasibility of time-triggered and event-triggered communication on a single networking infrastructure. Namely, TTEtherent permits the applications with varying timing constraints to exchange messages over the same physical network [77].

Figure 3.29 denotes the relationship between different layers of the OSI paradigm and TTEthernet. As Figure 3.29 shows, the packets from higher layers such as TCP and IP can be encapsulated as TTEtherent frames without any changes in packet's payload. The TTEthernet Protocol Control Frame (PCF) has its own format and is used to establish and keep the synchronized global time [76].

There are three different types of traffic in TTEtherent networks:

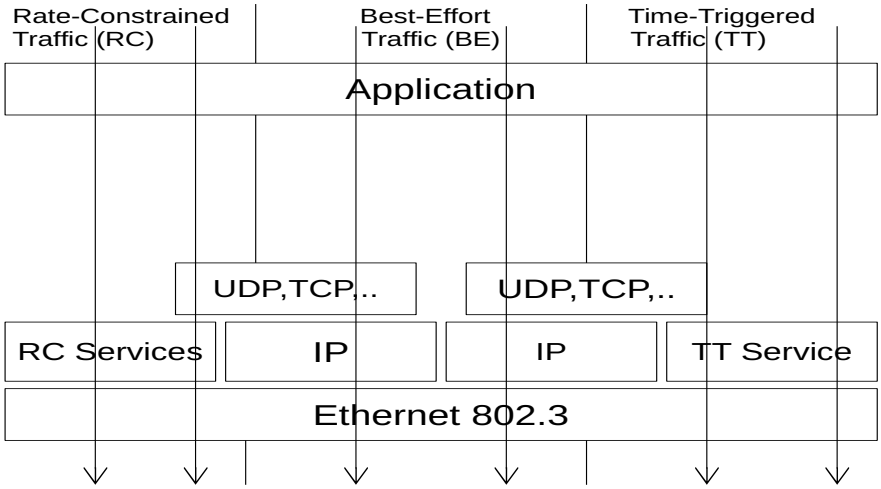


FIGURE 3.29: Relationship between different layers of OSI paradigm and TTEthernet [78]

- **Time-Triggered**: A sending end-system dispatches TT messages at pre-planned time slots where TT messages have collision-free access to the network resources such as physical links. As a result, the fixed transmission delay and low jitter are guaranteed for TT communication. The transmission schedules of TT messages are defined statically based on the network topology, and the specification of the data flows that are sent over the network. Therefore, any modification in a network such as adding a new data flow requires recalculation of transmission tables. This feature may limit the scalability of TT communication. However, this issue can be addressed through incremental scheduling schemes.

- **Rate-Constrained**: Rate-constrained (RC) communication which is defined in ARINC664 part 7 offers a bounded transmission delay through an adequate bandwidth allocation. Unlike TT communication, RC transfer does not require the global time base to limit latency. Instead, it specifies the minimum time interval between two successive messages belonging to the same RC flow (i.e. BAG). Therefore, if an end-system sends RC messages at a higher rate where the gap between successive message arrivals is less than the flow's BAG, messages will be dropped by a receiver. The RC transfer, however, does not rely on the global clock for the guaranteed performance, but it requires the computation of the upper bound of the transmission delay based on the network characteristics. Therefore, the parameters of an RC transfer (e.g. BAG and maximum end-to-end delay) must be recalculated in case of a network modification.

- **Best-Effort**: Unlike TT and RC communication, this class of traffic does not have timing requirements such as bounded latency and low jitter. Instead, the transmission of BE frames follows the best-effort communication paradigm, meaning that a BE message is served when the required network resources are available; otherwise, it is dropped. Consequently, the BE communication has a non-deterministic transmission delay and packet loss rate [78].

Transparent Synchronization. TTEthernet is also known as a transparent synchronization standard since it allows event-triggered (e.g. RC and BE messages) and time-triggered communication on the same physical network. In a TTEthernet network, the PCF messages which are used to transfer control data such as timing information have the highest class of service, meaning that they are the first ones to be served by network components even in the presence of other traffic types (e.g. TT and RC traffic).

Due to the distributed and fault-tolerant nature of the TTEthernet synchronization mechanism, multiple devices act as PCF generators. The PCFs are delivered to receivers over a multi-hop switched network. Thereby, they are received at destinations with the transmission delays corresponding to their forwarding routes. For this reason, each device that participates in the PCF delivery puts the transmission delay in a particular field of the PCF before dissemination.

The dispatch order of PCFs has high importance in the synchronization process, but it is quite likely that PCFs are not received in the same order as their dispatch sequence. The devices calculate a parameter called the permanence point in time to reconstruct the dispatch order of PCFs. The static maximum transmission delay of a PCF is the first ingredient of the calculation of the permanence point in time. After that, the permanence point in time can be calculated as follows:

$$\text{Permanence Point in Time} = \text{Receive point in time} + \text{Actual Transmission Delay} \tag{3.11}$$

The permanence point in time permits retrieving the local clock of other devices as well as the dispatch order of PCFs [76].

Clock Synchronization The TTEthernet clock synchronization is implemented in two steps. First, the devices with the role of Synchronization Master (SM) send Integration Frames (IN) which are a specific type of PCF towards the Compression Masters (CM). As a next step, the CMs forge and forward a new IN frame with the average value of the relative arrival times of incoming IN frames. The role of each device is selected based on the system design specification [78]. The simplified process of the TTEthernet synchronization is shown in Figure 3.30.

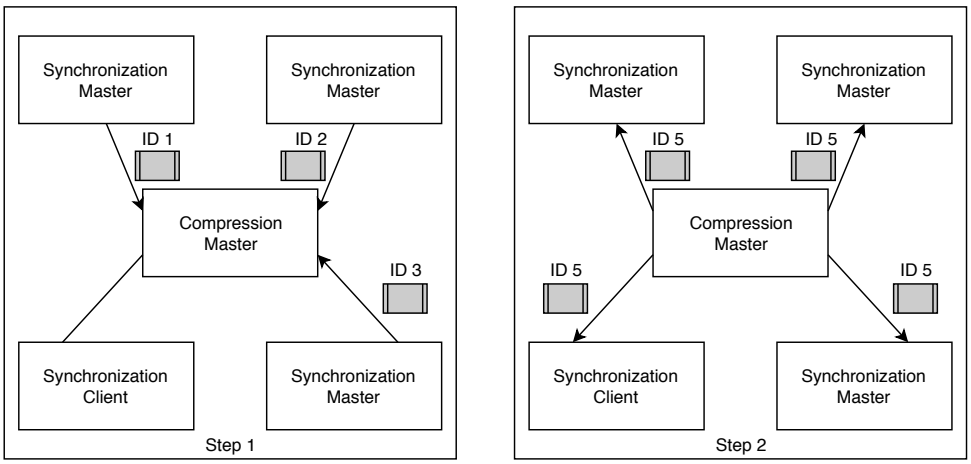


FIGURE 3.30: Simplified TTEthernet two-step synchronization mechanism [78]

TTEthernet Startup/Restart. For either start or restart of a TTEthernet network, an SM first dispatches a certain type of PCF messages that is called Coldstart frame (CS) when it is unable to synchronize to the global time within the predefined time interval. The CMs then broadcast the received CS frames to SMs except for the sender of the CS frame. An SM, upon reception of the CS, first waits for the predefined time interval and then sends a Coldstart Acknowledge frame (CA) to the CMs. As a next step, the CMs broadcast the CA frame to other SMs. This process is concluded by initiating the synchronization procedure from the SMs upon reception of the CA frame and after the fixed waiting, duration [78].

Scalable Fault-Tolerance For fault-tolerant purposes, the TTEthernet synchronization mechanism mitigates different types of faulty behaviours through active redundancy.

The faulty behaviour that may happen while devices communicate over the TTEthernet network can be categorized as follows:

- **Fail-Silence Failure Mode**: It happens when a device stops dispatching messages.
- **Fail-Omission Failure Mode**: It happens when a random number of messages are not sent/received by a device.
- **Fail-Inconsistent Failure Mode**: It happens when a message from a certain sender is received as a valid packet by some receivers whereas it is recognized as a wrong message by other receivers.
- **Fail-Inconsistent-Omission Failure Mode**: It happens when a faulty device perceives some incoming messages as invalid and other received frames as valid although all those messages are, in fact, correct.
- **Fail-Arbitrary Failure Mode**: It happens when a device dispatches messages from random egress ports at arbitrary time slots and with arbitrary payload.

TTEthernet tackles the mentioned failure modes via two failure hypotheses:

1. Single-Failure Hypothesis: TTEthernet network only masks a single failure, either an end-system with a fail-arbitrary failure or a switch with a fail-inconsistent-omission failure.
2. Dual-Failure Hypothesis: TTEthernet network masks two fail-inconsistent-omission failures in any two devices, including end-systems and switches.

It is noteworthy that TTEthernet switches act as fault-containment boundaries. The TTEthernet switches, in addition, may contain a central bus guardian function that changes the faulty behaviour of a set of end-systems from the fail-arbitrary to the inconsistent-omission failure mode and thus alleviates the effect of these faulty end-systems. Furthermore, TTEthernet addresses an inconsistent failure through two disjoint forwarding paths rather than three independent routes [76].

3.7.3 Audio Video Bridging Protocol

Over the last decades, Ethernet technologies have been widely deployed in a wide range of networks such as LANs and WANs since they fulfill different needs of various stakeholders from the high demand for bandwidth to the seamless connectivity

between the vendor-specific devices [79]. Ethernet due its maturity and efficiency is seen as a promising communication infrastructure for real-time systems. However, it does not offer temporal properties that are essential for real-time systems. To benefit from the Ethernet advantages, the Audio/Video Bridging (AVB) [13] task group introduced a series of protocol extensions to the IEEE 802.1 Ethernet standard which offers deterministic transmission of Audio and Video (AV) traffic over Ethernet networks [80].

The AVB standard defines the following procedures to achieve bounded latency and low jitter transmission of AV traffic:

- **IEEE 802.1AS**: is used to synchronize time-sensitive applications. This standard is thoroughly described in Section 3.5.5.
- **IEEE 802.1Qat [81]**: is used to set up forwarding paths for AV streams through resource allocation.
- **IEEE 802.1Qav [82]**: specifies AVB queuing and scheduling schemes.
- **IEEE 802.1BA [83]**: includes three IEEE 802.1 protocols. These protocols determine the number of AVB procedures and configuration profiles [84].

3.7.3.1 Stream Reservation Protocol

IEEE 802.1Qat is also known as Stream Reservation Protocol (SRP) and enables on-demand resource allocation for a certain stream across a bridged LAN. This protocol defines methods that are used to identify the necessary resources by AV streams and maintain the reserved resources during AV traffic transmission. In short, SRP allows the establishment of AV streams through registration and de-registration.

SRP uses three different signalling protocols for end-to-end stream establishment: Multiple MAC Registration Protocol [85] (MMRP) which is used for registration of talkers communicating over an Ethernet LAN, Multiple VLAN Registration Protocol [85] (MVRP) which is used for registration of tagged streams and Multiple Stream Registration Protocol (MSRP) which is used by talkers and listeners for signaling purposes. MSRP facilitates registration and de-registration of AV streams in a similar way to MMRP and MVRP, but it also allows dynamic modifications of registered information, unlike MMRP and MVRP.

MSRP uses five types of declaration (i.e. Talker Advertise, Talker Failed, Listener Ready, Listener Ready Failed and Listener Asking Failed.) for end-to-end stream establishment. A talker announces the specification of the stream that it aims to dispatch through a Talker Advertise declaration. The bridges, upon reception of a Talker Advertise declaration, register the associated information and forward the declaration to other nodes. On the other hand, the Talker Advertise declaration that arrives at a listener will be registered. A listener replies this declaration with a Listener Ready declaration if it wants to receive the declared stream. The bridge utilizes the StreamID embedded in the Listener Ready declaration to find the port where the associated Talker Advertise is registered. Then the bridge sends back the Listener Ready declaration to the Talker from that port while it allocates the necessary resources for the declared stream and adjusts its filtering and queuing schemes accordingly.

When any Bridge between a talker and a listener can not allocate the required resources (e.g. bandwidth) defined in the Talker Advertise declaration, it forwards a Talker Failed instead of the Talker Advertise declaration. Likewise, if there are not

enough resources for the stream, a Listener Asking Failed declaration is sent back to the Talker instead of the Listener Ready declaration.

The Listener Ready Failed declaration is used where at least one of the listeners informs the Talker that it can not provide the required resources through a Listener Asking Failed declaration while other listeners send back a Listener Ready meaning that it has the necessary resources for the stream. The bridge which receives Listener Asking Failed and Listener Ready declarations on different ports will forward a Listener Ready Failed to the Talker.

It is noteworthy that MSRP is also used to transport Stream Reservation (SR) class specifications among stations in bridged LANs. The nodes which share the same SR class specification form an SRP domain [81].

3.7.3.2 Forwarding and Queuing Enhancements for Time-Sensitive Streams

IEEE 802.1Qav is also known as Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQETS) and specifies shaping and queuing schemes for AVB streams so that AVB-aware nodes can guarantee bounded end-to-end latency and low jitter transmissions of AV streams. More specifically, this standard defines the classification of AVB streams and also QoS characteristics for each class of AVB stream. In addition, FQETS shapes traffic in a way that AVB traffic which has reserved resources prioritizes over non-AVB traffic [82].

In AVB networks, traffic can be classified as follows:

- **SR class A**: is associated with the highest priority traffic in AVB networks. The transmission of SR class A traffic over seven hops may take at worst two milliseconds. To measure end-to-end latency, MSRP utilizes timing information, including the residence time and the link delay provided by IEEE 802.1AS.
- **SR class B**: is associated with the second-highest priority traffic in AVB networks. The transmission of SR class B traffic over seven hops may take at worst 50 milliseconds.
- **Best-effort traffic**: is associated with the lowest priority traffic in AVB networks. Any traffic that does not belong to the AVB SR classes, is classified as best-effort traffic.

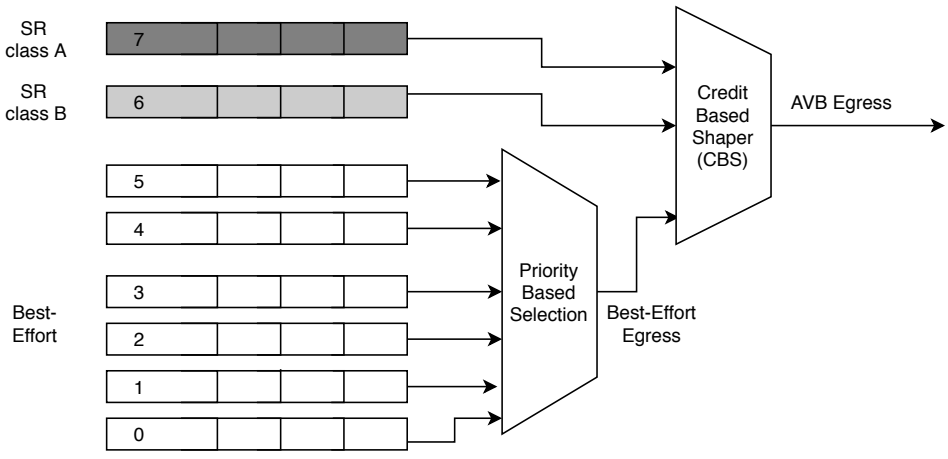


FIGURE 3.31: An example of a queuing scheme in an AVB-aware device [86]

At each port, each traffic class has its dedicated queues. Figure 3.31 denotes an example of a queuing scheme in an AVB-aware device. Therefore, upon reception of traffic, frames are put into different egress queues based on their priority values that are embedded in the Class-of-Service (CoS) field. It is quite likely that traffic arriving from non-AVB networks has a CoS field that maps to the AVB SR classes, while representing non-AVB traffic. The priority value of every frame is regenerated to mitigate this issue, before transmitting traffic over AVB networks. After enqueuing frames, a transmission selection algorithm is used to decide which egress queue is eligible to send out a frame [84].

Each egress port employs the credit-based shaper algorithm to shape outgoing traffic so that burstiness of AVB traffic does not lead to the starvation of lower priority traffic (i.e. best-effort traffic). Using the credit-based shaper, frames associated with a specific stream are placed into the queue serving the stream's traffic class. The enqueuing process is performed based on idleSlope. The idleSlope defines the maximum percentage of a port's bandwidth which is assigned to a particular queue associated with the specific traffic class. Therefore, the idleSlope specifies at which rate frames are put into the correct egress queue. It is noteworthy that the rate of enqueuing frames associated with a specific stream, can not exceed the reserved bandwidth for that stream. On the other hand, the credit-based shaper limits the transmission rate of frames residing in the queue, which is selected by the transmission selection algorithm based on sendSlop. SendSlop determines the rate at which the credit decreases while frames are sent out from egress queues.

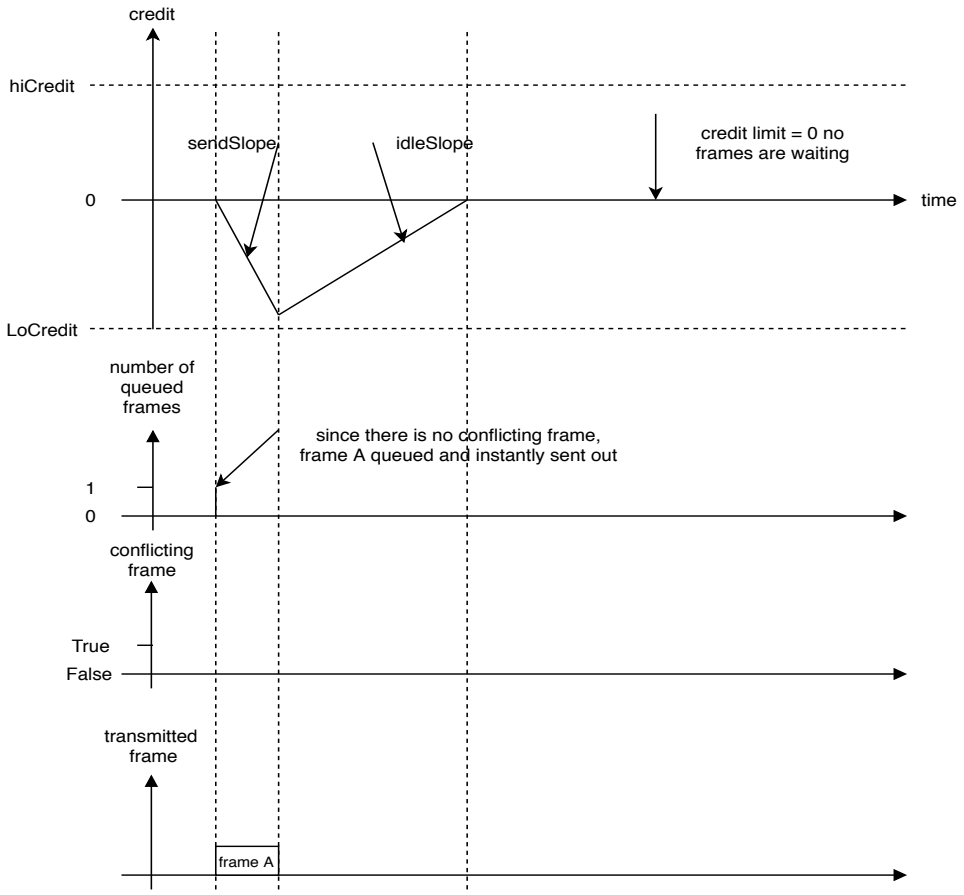


FIGURE 3.32: Credit-based shaper operation-Scenario 1 no conflicting traffic [82]

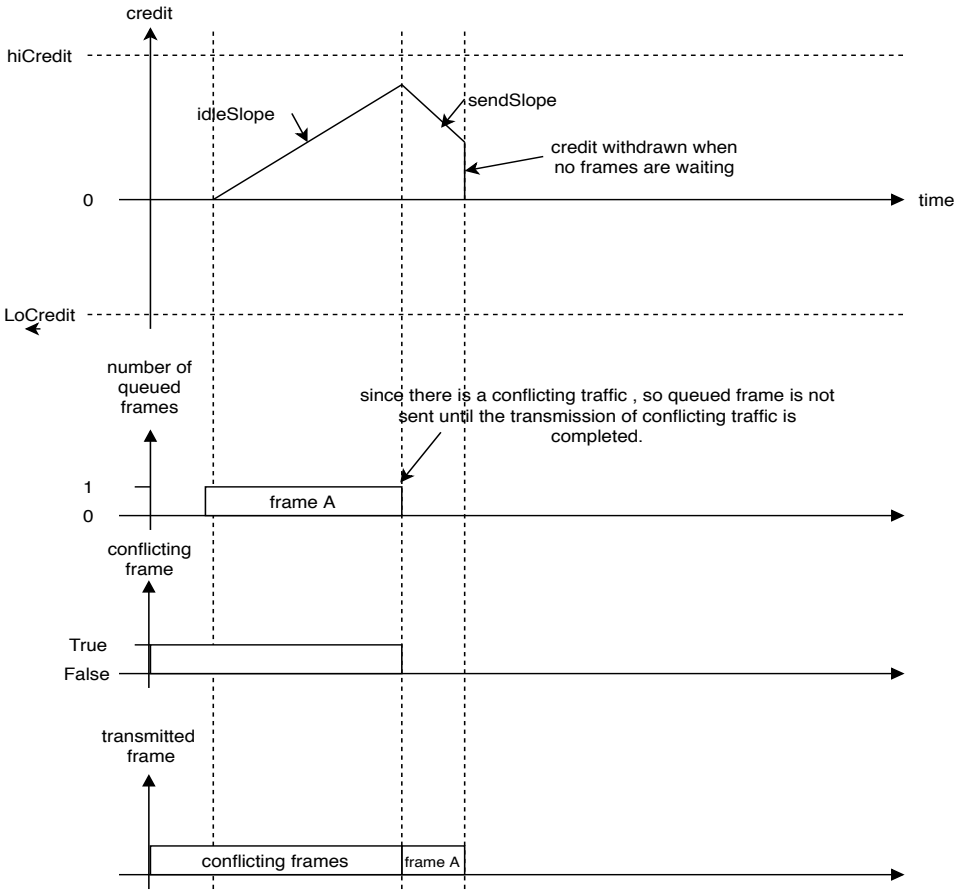


FIGURE 3.33: Credit-based shaper operation-Scenario 2 with conflicting traffic [82]

Figures 3.32-3.34 depict how the credit-based shaper operates. In the first scenario, it is assumed that a transmission decision is granted for an AVB frame as soon as it is placed into the egress queue since the higher priority queues are empty. As shown in Figure 3.32, the credit decreases with respect to sendSlope while the frame is sent out. The credit recovers to zero at the rate of idleSlope after finishing the frame transmission so that other frames can be sent out. In the second scenario, an AVB frame arrives at the egress queue when another frame is in transmission. As shown in Figure 3.33, the credit increases with respect to idleSlope until the port grants transmission for the AVB frame. The accumulated credit decreases during the transmission of the AVB frame. Since the credit is more than the amount required by the AVB frame and there are no other frames waiting for transmission, the credit decreases to zero after finishing the frame transmission. In the last scenario, as illustrated in Figure 3.34, three AVB frames are placed into the egress queue, and the credit increases with respect to idleSlope until another frame is in transmission. The port starts transmitting AVB frames as soon as the transmission of the conflicting frame is completed. Since the first two frames consume all accumulated credit, the transmission of the third frame starts when the credit recovers to zero [82].

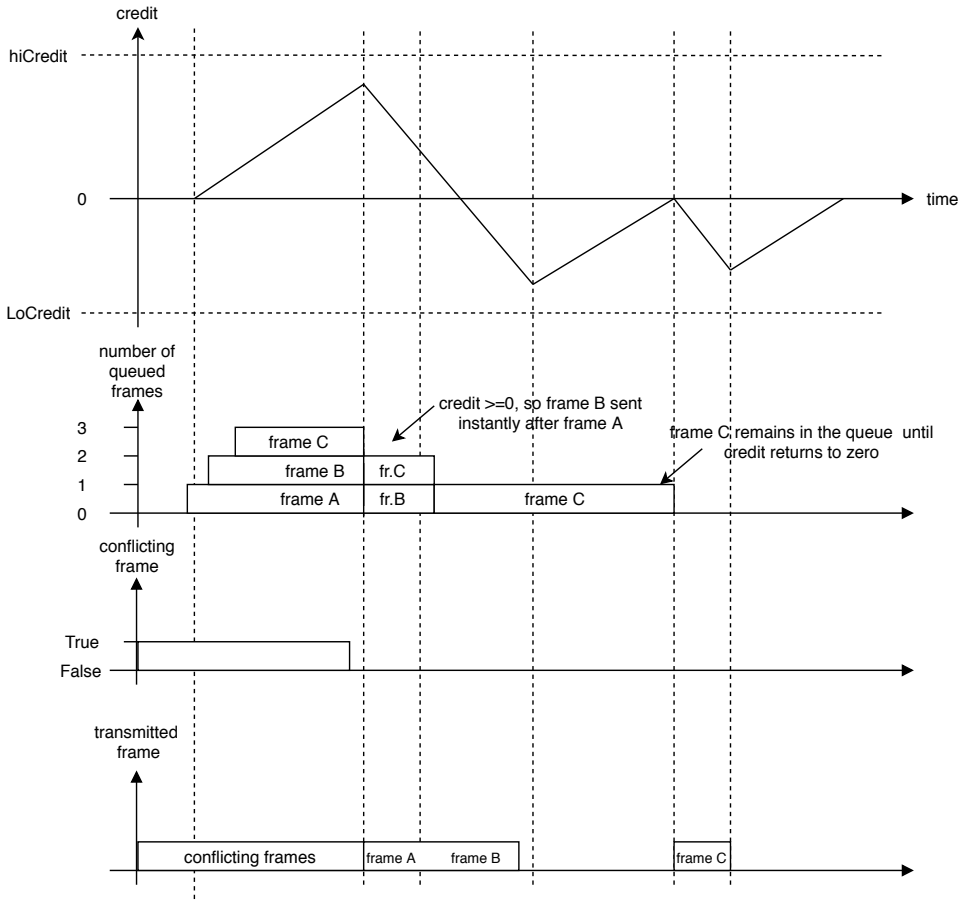


FIGURE 3.34: Credit-based shaper operation-Scenario 3 with burst traffic [82]

3.7.4 Time-Sensitive Networking

Highly reliable, scalable and deployable networks with strict temporal constraints are inevitable for modern cyber-physical systems. Due to widespread usage and success of Ethernet technologies, the Time Sensitive Networking [11] task group introduces a series of protocol extensions to the IEEE 802.1 Ethernet standard [43]. These standards provide real-time capabilities and performance improvements. The convergence of synchronous, asynchronous and best-effort traffic on a single network is the key aspect of TSN. TSN standards are built on top of the Audio Video Bridging (AVB) protocol suites [13]. AVB is specified to provide guaranteed latency and fixed jitter for the audio and video transmission by reserving bandwidth throughout the path from a sender to a receiver. Despite the success and widespread use of AVB in automotive networks, AVB is not able to fulfill the requirements of mission-critical applications like strict timing constraints [87].

The main goal of TSN is to focus on the uncovered areas in AVB sub-standards. To achieve this, the TSN task group develops a fault-tolerant synchronization mechanisms, a time-sensitive transport protocol and enhancement mechanisms for the Stream Reservation Protocol. The task group also introduces a robust redundancy procedure to prevent traffic loss in case of any failure at the different levels of the network. Furthermore, TSN includes time-aware scheduling and policing mechanisms. The features as mentioned above lead TSN to be a real-time capable, reliable and interoperable standard, which is suitable for different industrial automation and

control networks (e.g. railway, avionics and automotive).

3.7.4.1 Enhancements for Scheduled Traffic

The Credit-Based Shaping (CBS) mechanism of AVB is unable to fulfill timing constraints of Time-Triggered (TT) streams, while time-insensitive traffic is transported over the same physical infrastructure. The CBS is applied to a minimum of two AVB traffic classes (i.e. SR class A and B) to avoid burstiness of AVB streams and starvation of lower priority traffic. This scheduling scheme is non-preemptive, which means the lower priority traffic can interfere with TT flows which have strict timing requirements and block their transmission. Therefore, CBS is unable to offer deterministic end-to-end latency and tight jitter for the mission-critical applications [87].

Time Aware Shaping (TAS) introduced in IEEE 802.1Qbv [12] is developed to resolve these problems of the AVB shaping mechanism. TAS is a preemptive scheduling method in which scheduled traffic always preempts lower priority traffic in order to meet its transmission schedule. In non-deterministic networks, delivery delay is estimated using analytical methods like network calculus [88, 89, 90, 91] and the trajectory method [92, 93, 94, 95]. However, transmission schedules for TT traffic in TSN networks are computed offline based on the network topology and the TT stream characteristics [79]. Thereby, TSN uses a centralized control and configuration mechanism to solve the scheduling problem, which is different from the hop-by-hop and decentralized reservation approach of AVB.

Similar to regular Ethernet devices, each port of a TSN-aware device has a number of queues proportional to the number of supported priorities. The higher priority queues are considered as TT queues, and the remaining queues follow the same queuing scheme as the AVB-aware devices. It means that few queues are assigned to AVB classes depending on the number of AVB stream classes supported by the AVB-aware device. The rest is dedicated to the Best Effort (BE) traffic that does not have any strict temporal constraints [96].

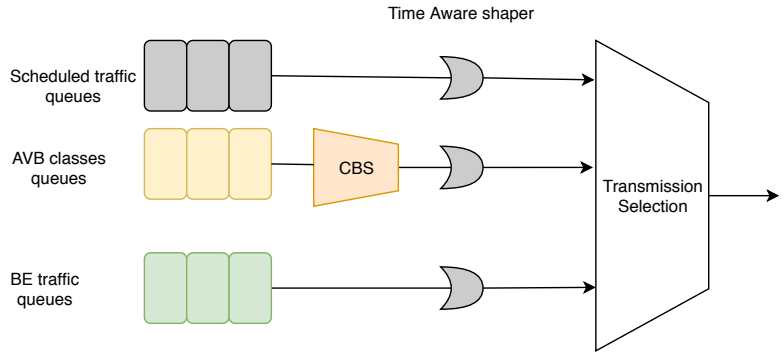


FIGURE 3.35: Queuing and scheduling scheme of an 802.1Qbv-aware device [12]

The time-aware shaper is defined based on the Gate Control List (GCL) concept. In this approach, all TT flows are placed into the queues dedicated to TT traffic and the GCL is applied to egress port queues unlike other TT protocols (e.g. TTEthernet) that place each TT flow in a separate buffer and apply a schedule to each buffer according to its requirements. This is the main reason TAS requires knowledge of the device queue configuration in addition to requirements of TT streams. The GCL is specified for each egress port and defines at each instant of time which queue is

eligible to transmit traffic. TSN extends the set of traffic types in AVB with an additional TT traffic type. This traffic type is specified for applications that have strict timing requirements, and despite AVB stream classes do not allow interference with less demanding applications. The TT traffic type is a prerequisite for deterministic networks with hard real-time applications, and it is transmitted periodically. Therefore, each flow in TSN networks is assigned to one of the following types: TT traffic, AVB classes or Best Effort (BE) traffic. In TSN, the traffic type assignment strictly depends on 802.1Qbv-capable switch configurations that specify the characteristics of incoming TT streams [79].

In an 802.1Qbv-capable device, first, each incoming frame is classified as a specific traffic type using different priority metrics (e.g. Priority Code Point (PCP) in 802.1Q header) and then based on its traffic type it is placed into the correct egress port's queue.

At each egress port, to select a frame for transmission, first, the queues whose gates are enabled are specified using GCL. If more than one queue is eligible to transmit traffic, a queuing scheme and transmission selection algorithm like CBS specifies which queue can transmit a frame. It has to be noted that when the gate of a TT queue is open, all other queues must be blocked to provide temporal isolation and deterministic latency for the scheduled traffic. In contrast, the gates of AVB and BE queues can be enabled at the same time, since these traffic types do not require freedom of interference from other queues. A TSN-aware device acts as a standard Ethernet element if the gates of all queues are open permanently. Figure 3.35 presents the queuing configuration and the scheduling scheme of an 802.1Qbv capable device.

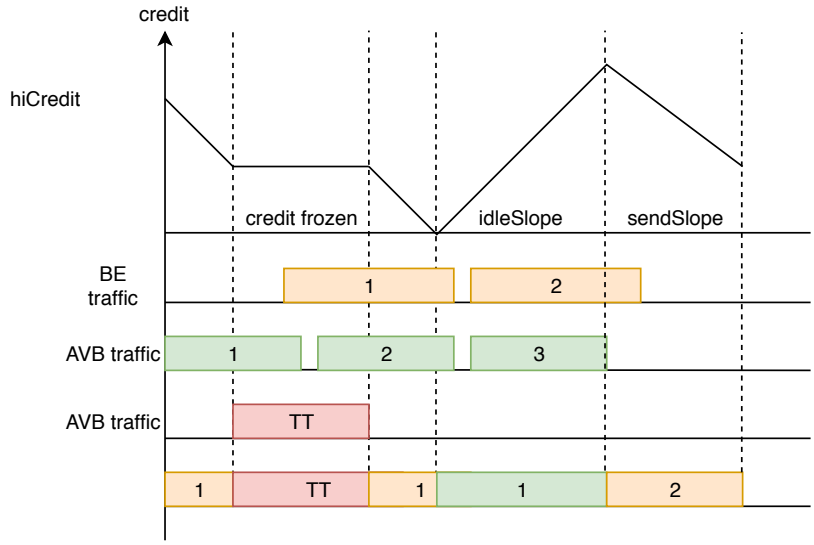


FIGURE 3.36: An example of transmission scheduling in an 802.1Qbv-aware switch [79]

For a better understanding of TAS operation, Figure 3.36 illustrates a scenario in which the 802.1Qbv-capable switch receives the AVB traffic for a while. While the AVB frames are still arriving at the switch, it starts receiving a TT flow and BE traffic simultaneously from different ingress ports but destined to the same egress port. The AVB stream is also directed to the same egress port as the TT and BE traffic. As Figure 3.36 depicts, the AVB frames would be sent out until the time slot scheduled for the TT frames starts. The transmission of the AVB stream is possible when the

control gate of the AVB queue is open based on the GCL, and the queue's credit is zero or positive according to CBS.

As shown in Figure 3.36, at t_0, the credit of AVB queue is positive, and the transmission of AVB frame 1 starts. The transmission of AVB frame 1 stops at t_1 as soon as the TT time slot starts. The credits of the AVB queue are frozen during the TT transmission window. However, the reception of AVB frames continues during the TT scheduled slot. The transmission of AVB frame 1 resumes at t_2 while the BE traffic is waiting for transmission in the lower priority queue. At t_3, the transmission of AVB frame one finishes but the credits of the AVB queue become negative. Therefore, the AVB frame two, which was enqueued during the TT time slot, cannot be transmitted. In addition, no TT flow is scheduled for the time slot starting at t_3. Hence, BE traffic transmission starts while the AVB queue credits are accumulating. The transmission of BE frame 1 completes at t_4 and at the same time, the transmission of AVB frame 2 starts [79].

3.7.4.2 Time-Based Ingress Policing

In fully deterministic networks, each device must know the arrival time of TT flows at ingress ports so that it can police and transmit frames based on the predefined schedule tables. The TSN task group develops IEEE 802.1Qci [25] to achieve this goal. The property, as mentioned earlier, is addressed with the time aware Access Control List (ACL) and ingress policing. The time-based ACL grants the pass/fail, Maximum Transmission Unit (MTU) size and target queue decision for each incoming TT frame at each instant of time. An 802.1Qci-capable device uses time as a correctness criterion for filtering and policing TT traffic. Same as the predefined TT schedule tables in IEEE 802.1Qbv, the time aware ACL is also defined offline. The time-based ACL must also be aligned with the GCLs of Ethernet ports. For instance, a TT frame belonging to a particular TT stream could be successfully transmitted, if the time aware ACL grants pass permission, and at the same time there is an allocated time slot for the frame in the GCL of the egress port.

The key benefit of time-based ingress policing is to protect an 802.1Qci capable device from a wide range of network attacks like man-in-the-middle attacks and babbling idiot faults. This sub-protocol makes the device more robust by blocking TT frames arriving outside their scheduled windows. Therefore, the possibility of sending TT frames in arbitrary order can be eliminated. It also results in the optimized usage of network and switch resources like link bandwidth and memories [97].

3.7.4.3 Redundancy Management in TSN

The TSN task group introduces the IEEE 802.1CB [28] standard to improve robustness and reliability of stream transmissions, especially for safety-critical traffic. A sender or a relay system (e.g. switch) with FRER capability, first generates and encodes a sequence number for each outgoing frame. Then it forwards the multiple copies of the frames towards the destination over multiple routes. Hence, in case of any failure, the frame is delivered to the destination via the redundant path. Therefore, the FRER mechanism decreases the probability of traffic loss considerably. In TSN, the IEEE 802.1Qca [98] protocol is deployed to configure the alternative routes for each stream. Besides, to avoid network overloading, the duplicated frames are eliminated either at intermediate relay systems or at a receiver.

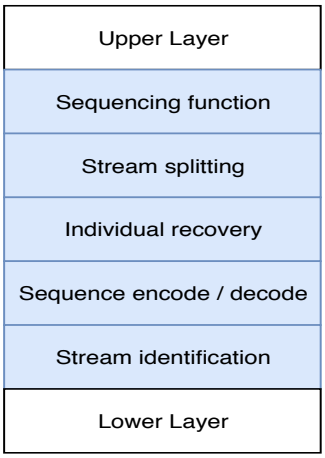


FIGURE 3.37: FRER functions

As shown in Figure 3.37, FRER consists of five different functions. Depending on the packet processing direction in a FRER-capable device, each function acts differently.

Sequencing. This module generates a sequence number for every frame passed down from the upper layer to the physical layer. In contrast, for each frame passed up in the protocol stack, the sequencing function examines the sequence number of a frame and discards the duplicated frames whose copies have been received before.

Stream Splitting Function. This function makes zero, one or more copies of every frame passed down to the physical layer according to the stream split table. Each copy of a frame will be transmitted to the destination via separate paths.

Individual Recovery Function. It checks the sequence number of every frame passed up in the protocol stack and eliminates the frames whose duplicates have been received previously. This function and sequencing provide similar services, but individual recovery functions can apply to multiple ports while the sequencing module is port specific. It is good to note that the recovery module will process a frame only if the integrity of the frame is verified by the lower layer (e.g. physical layer) validity checks.

Sequence Encode/Decode. This function encodes a sequence number generated by the sequencing function into the frame passed down for transmission. In the reception direction, the function derives the sequence number from the frame passed up in the protocol stack. FRER introduces the Redundancy tag (R-TAG) as an example of sequence number formatting. R-TAG comprises of three parts: 1) EtherType which has the value F1C1. 2) Reserved field which occupies the second two octets of R-TAG and is reserved for future revisions of IEEE 802.1CB. 3) Sequence number field which is encoded in the last two octets of R-TAG. Figure 3.38 illustrates the format of the R-TAG.

The encoding mechanism must be known to all relay systems and end-systems. Otherwise, they cannot decode the sequence number from incoming frames.

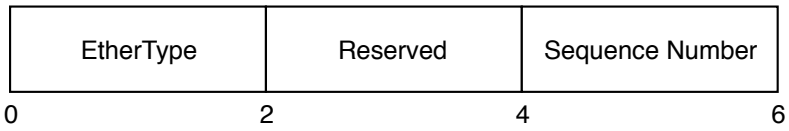


FIGURE 3.38: R-TAG header structure

Stream Identification Function This function specifies a stream identifier for every frame received either from the physical layer or the upper layer. The stream identifier determines to which stream the frame belongs and how it should be processed by other FRER functions. The four different stream identification approaches are listed in Table 3.2. The stream identification functions can also overwrite some of the parameters of the frame's header to reflect the stream identifier.

Stream Identification functions	Stream Identifier
Null Stream identification	Dst MAC address, VLAN ID
Source MAC and VLAN	Src MAC address, VLAN ID
Destination MAC and VLAN	Dst MAC address, VLAN ID
IP octuple	Dst MAC address, VLAN ID IP src address, IP dst address IP next protocol, src port, dst port

TABLE 3.2: Stream Identification Functions

Depending on the network design, the FRER functions can be placed in the protocol stack of the egress port in different orders. To be more specific, each port of a device selects different FRER functionalities. Due to this flexible design, the FRER-capable devices can inter-operate with network elements that are unaware of FRER. Furthermore, FRER protects TSN networks from faulty behaviours like a stuck transmitter. To detect such errors, a network element with FRER capability saves the history of a stream's sequence numbers. Using this information, it discards frames with a sequence number different from the expected value [28].

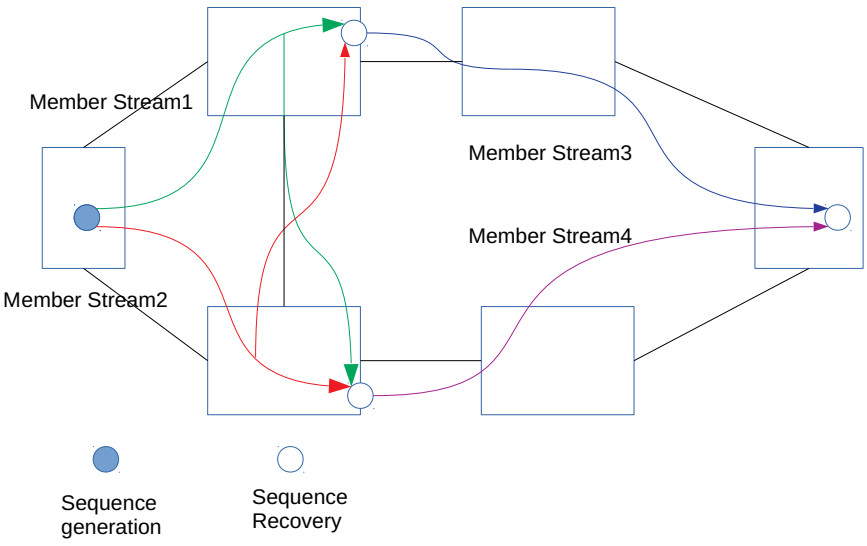


FIGURE 3.39: Sequence generation and recovery functions [28]

Figure 3.39 depicts how FRER operates. The sender (i.e. the leftmost box) generates a sequence number and embeds that into the outgoing frames. At the next nodes where two copies of the frames are received, the duplicate frames are eliminated while the first copy of the frames is forwarded. The receiver (i.e. the rightmost box) passes up the first copy of the frame to the higher layers in the protocol stack. However, it discards the duplicate frame. In this setup, frames are delivered to the receiver regardless of any single link failures.

3.7.4.4 Stream Reservation Protocol Enhancements and Performance Improvements

IEEE 802.1Qcc [26] introduces improvements to mechanisms and managed objects which were used to reserve network resources for time-sensitive applications. For configuration of TSN features, first talkers and listeners exchange their specification and then all bridges that are part of the forwarding routes between talkers and listeners are configured accordingly.

The User / Network Interface (UNI) is one of the significant pillars in TSN configuration models. The talkers and listeners reside at the user side of the interface. On the other hand, the bridges that participate in the forwarding frames are on the network side of the interface. Every talker declares its stream characteristics to the network. On the other hand, the network configures the bridges based on its knowledge about the network structure and the stream specifications. The network also notifies the user about the status of the establishment of the stream.

There are three different configuration models to configure TSN features in users and bridges:

- **Fully distributed model**: Each Talker distributes its stream specifications to other network nodes. Since there is no centralized network configuration element in this model, the configuration information is directly forwarded to the bridges that are part of the forwarding path for the given stream. Every bridge configures its resources using the received information which may not include the details about the whole network. This configuration model can be used to set up parameters associated with the credit-based shaper.

- **Centralized network / distributed user mode**: Talkers declare their stream characteristics to the network. However, in this model, unlike the fully distributed model, the user requirements are sent to a Centralized Network Configuration (CNC) element for further processing. The CNC also gathers information on network structure and bridge capabilities through a network management protocol (e.g. NETCONF). Therefore, the CNC knows the entire network and can use this information for the configuration of TSN features such as time-aware shaper, frame preemption and FRER. In short, the CNC facilitates computation of TSN features which require more resources compared to a model where all bridges participate in the configuration process. After computing the configuration parameters, the CNC configures the bridges using a network management protocol.

  To use network resources more efficiently, the bridges that are attached directly to the end stations direct the user requirements to the CNC instead of forwarding the information within the network.

- **Fully centralized model**: A Centralized User Configuration (CUC) element first gathers information on user capabilities and requirements and then configures end stations accordingly. This is different from the above models where talkers and listeners exchange the configuration information over the network. However, the fully centralized model in a similar way to the models above uses the CNC to obtain bridge capabilities and configure the bridges. This configuration model, due to its architecture is beneficial for TSN networks where end stations are configured using a vast number of parameters [26].

In TSN networks, configuration data is encoded using either YANG [99] or TLV fields and is transported through different management protocols such as NETCONF [100].

3.7.4.5 YANG

YANG (Yet Another Next Generation) [99] aims to model data which is transported over the network using network management protocols, especially NETCONF. Namely, YANG specifies data models for different operations of network management protocols such as state data, configuration data and notifications. In YANG, the hierarchical data is modelled as a tree where each node may have specific conditions for its existence or a set of values that it can get. Additionally, YANG defines data models as either modules or submodules where modules may consist of definitions from other submodules or external modules. Each YANG module defines a namespace which acts as a Uniform Resource Identifier (URI) [101].

In YANG, apart from the built-in types and standard modules, user-defined types and extensions can also be added. Moreover, the nodes in the hierarchy with similar characteristics can be grouped [99]. Since data used by the network management protocols is encoded in XML or JSON, a YANG module that contains the data model of the network management features can be converted to XML in a way that all nodes engaged in configuration process understand each other accurately [26].

An example of a YANG module that corresponds to a NETCONF event notification and its XML representation are presented in Figure 3.40 and 3.41 respectively. As shown in Figure 3.40, the event module specifies a namespace which is used in the XML representation for referencing the module [102].

```
module event {
  namespace "http://example.com/event";
  prefix "ev";
  container event {
    leaf event-class {
      type string;
    }
    anyxml reporting-entity;
    leaf severity {
      type string;
    }
  }
}
```

FIGURE 3.40: A YANG model for event notification

```
<notification xmlns="
   urn:ietf:params:netconf:capability:notification:1
   .0">
  <eventTime>2019-05-17T10:00:00Z</eventTime>
  <event xmlns="eti.uni-siegen.de:es:yang:tsn">
     <event-class>topology change</event-class>
     <reporting-entity>
        <device>switch_1</device>
     </reporting-entity>
     <severity>major</severity>
  </event>
</notification>
```

FIGURE 3.41: An example of NETCONF notification

3.7.4.6 Network Configuration Protocol

The Network Configuration Protocol (NETCONF) [100] determines procedures for deployment, modification and removal of configuration information on network elements. All types of NETCONF messages, including configuration data and state data, are encoded Extensible Markup Language (XML) [103]. In NETCONF, a network manager acts as a client while other network devices (e.g. bridges) act as servers. Furthermore, the NETCONF operations are modelled through Remote Procedure Calls (RPCs) [104]. Thereby, a client and a server which are connected via a NETCONF session implement NETCONF operations using an RPC-based method.

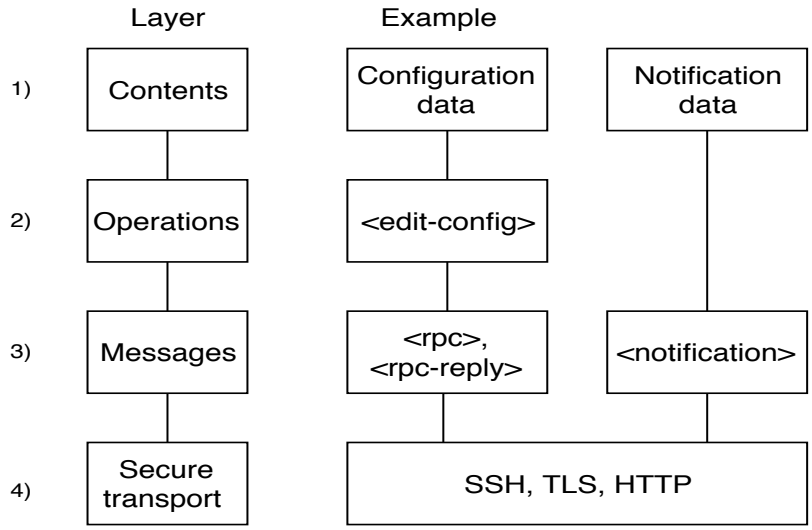


FIGURE 3.42: NETCONF protocol layers [100]

As Figure 3.42 illustrates, there are four different layers in the NETCONF architecture:

1. **Secure Transport layer**: is used to transport NETCONF messages among network devices. NETCONF, however, can use any transport protocol for this layer, the selected transport standard must meet the primary requirements.

2. **Messages layer**: offers a transport-independent encapsulation for messages carrying notifications, RPC requests and responses.
3. **Operations layer**: includes the definition of NETCONF operations and associated input parameters.
4. **Content layer**: uses other protocols (e.g. YANG) as a data modelling language.

In a NETCONF system, there are two types of information: configuration data which is used to configure the system from the initial state and state data, which consists of statistics and status data. The main reason to differentiate configuration data from state data is to avoid transferring unnecessary information during the configuration process.

RPC Models. In NETCONF, a client invokes a certain operation on a server by issuing an RPC request message. A server, in response to the RPC request message, returns an appropriate RPC reply message. In short, NETCONF follows RPC-based communication principles where NETCONF request and reply messages are exchanged between a client and a server. NETCONF requests and replies are encapsulated using <rpc> and <rpc-reply> elements. Since each <rpc> element and corresponding <rpc-reply> element has the same identifier (i.e. "message-id" field), a client can correlate the RPC request message to the corresponding RPC response message.

Configuration Datastore. A configuration datastore contains all necessary configuration data which is used to transform a network device from the initial state to the running state. The configuration datastore can be specified for different states of the system (e.g. startup and running). However, a device is only obligated to have a running datastore that contains its current configuration data.

Operation. NETCONF introduces a limited number of operations to facilitate configuration/reconfiguration of a network device and also retrieval of the device's state data. The primary NETCONF operations are:

- **get**: is used to retrieve the active configuration and state data of a server.
- **get-config**: is used to retrieve requested information from the device's configuration data.
- **edit-config**: is used to modify the server's configuration datastore based on given configuration information.
- **copy-config**: is used to replace the server's configuration datastore with a given configuration datastore. To perform this operation, a server generates a new datastore, if it does not have one.
- **delete-config**: is used to remove the server's configuration datastore. It has to be noted that this operation does not apply to the active configuration datastore.
- **lock**: is used by the client to acquire the lock of the server's configuration datastore for a specific interval so that other devices cannot access the datastore during that time.

- **unlock**: is used to release the acquired lock by the lock operation.
- **close-session**: is used to terminate an active NETCONF session and release all corresponding resources.
- **kill-session**: is used to abort an active NETCONF session and release all related resources.

Since the invocation of a NETCONF operation is not always successful, the client must check an RPC reply for the operation's output [100].

Chapter 4

Scheduling Strategies for Time-Sensitive Networking

In modern industrial systems, the increasing number of sensors and computing nodes results in complex network designs and a high volume of data exchanges over the communication links. Although Ethernet fulfills the high demand for bandwidth and the seamless connectivity of vendor-specific devices, it does not offer real-time capabilities which are essential for cyber-physical systems. Time-Sensitive Networking (TSN) [11] is a set of standards that introduces several Ethernet extensions and a novel scheduling mechanism called Time-Aware Shaper (TAS). TAS uses time as a correctness criterion instead of a metric for the performance measurements [16]. This feature enables TAS to provide determinism, low latency and low jitter communication for safety-critical applications.

In TSN networks, a fault-tolerant synchronization protocol (i.e. IEEE 802.1AS-rev [17]) ensures that all devices are synchronized to global time and provides the basis for deterministic TT communication. Besides, each port of a device dedicates some queues to the TT flows and the rest for non-TT communication. A TSN capable device transmits messages according to the Gate Control List (GCL). The GCL determines at each time instant which queue is eligible to send out messages. TAS applies the port's GCL with respect to global time. Therefore, it guarantees that the allocated time slot for a TT flow will not be occupied by any other message (including other TT flows and non-TT messages). The port-specific GCL along with a fault-tolerant clock synchronization enables Ethernet-based networks to meet strict timing constraints of mission-critical applications.

In TSN due to the complexity of the scheduling problem, the port-specific GCLs are computed off-line. Despite GCL advantages, the scheduling problem arising from the GCL synthesis is NP-complete. Thereby, it is challenging to develop scheduling strategies which can be employed in different networks with varying structures and sizes.

To synthesize GCLs, both knowledge of the network topology and the TT flow specifications are required. In modern cyber-physical systems, an ever-increasing number of network elements including end-systems, switches and links results in numerous potential forwarding routes and consequently a large number of scheduling possibilities for each TT flow. The feasibility of running real-time applications over different end-systems makes the search space of legitimate schedules even bigger. Therefore, the optimization algorithms for the search space exploration, which are a vital element for the deployment of TSN gain significant attention. Several works discuss the scheduling problem of time-triggered networks by reducing the complexity of the problem using several assumptions. For instance, the majority of TT scheduling solutions calculate the global schedules regardless of routing possibilities. In other words, they ignore the impact of routing on the scheduling constraints

and use fixed routes which are generated separately as an input to the schedulers. This simplified abstraction of the scheduling problem (e.g. using fixed routing) may lead to the failure of the schedule generation, although the system is schedulable [105]. A few recent works [105, 106] focus on joint routing and scheduling for the TT schedule computation. They all use ILP-based approaches, which are rather slow and not scalable to solve large real-time systems. Additionally, these solutions do not consider job scheduling, inter-flow dependencies and distributed real-time applications.

In this chapter, we present two different scheduling algorithms: the first one is based on a genetic algorithm, while the second one is based on list scheduling. In contrast to state-of-art scheduling solutions, the Genetic Algorithm (GA) and Heuristic List Scheduler (HLS) generate port-specific GCLs imposing routing and scheduling constraints at the same time. These solutions transform two separate sets of routing and scheduling constraints into one set of constraints and solve the scheduling problem considering interdependencies of TT flows in a single-step. The main goal of these schedulers (i.e. GA and HLS) is to satisfy the deadlines of TT traffic while optimizing the TT transmission makespan and the overhead of TT communication. To achieve this goal, they minimize the gap between TT time slots and consequently reduce the number of guard bands that are introduced before every TT time slot to avoid interference with other flows. Additionally, these solutions, aside from message scheduling, incorporate job scheduling and as a result, provides an opportunity to distribute safety-critical applications over available end-systems rather than placing them on a particular device. This approach is beneficial for mission-critical applications (e.g. driving assistance) that require considerable amounts of computational power.

To have a solid base for comparison, we also developed a basic list scheduler which is a typical example of state-of-art scheduling procedures and solves the routing and scheduling problems separately. To this end, a basic list scheduler first finds the shortest paths between all end-systems and then uses these fixed routes for solving the scheduling problem. In the experiment section, we apply the proposed solutions to different sets of TT flows and network designs. We also compare the simulation results of the single-step GA and HLS with the two-phase basic list scheduler. The results demonstrate that GA and HLS improve the scheduling capability and transmission efficiency of TT communication over the basic list scheduler considerably.

The remainder of the chapter is structured as follows: In Section 4.1, related work is discussed. Section 4.2 introduces the system model used in this work. Section 4.3 formulates the scheduling problem of TT communication. The joint routing and scheduling constraints are defined in Section 4.4. Section 4.5 describes the GA procedure while Section 4.6 details the heuristic list scheduler and two-phase list scheduler which serves as a reference implementation of existing scheduling solutions and also as a baseline for the experimental analysis. In the last section, experimental results are evaluated.

4.1 Related Work

Since TSN systems necessitate transmission schedules to guarantee temporal properties, in recent years, several works addressed the scheduling problem of TAS, which is also introduced in the IEEE 802.1Qbv [12] extension. However, before the introduction of TAS, the scheduling problem of time-triggered systems in the context

of different networking technologies was studied intensively. For instance, a significant number of scheduling strategies have been designed for TTEthernet networks. In [107], Steiner first specified the scheduling constraints for TTEthernet networks and then developed a Satisfiability Modulo Theory (SMT)-based solution to compute static TT schedule comprising fixed-size time slots overlooking the presence of other types of traffic (e.g. RC traffic). Steiner [108] also implemented another SMT-based solver for scheduling of TT communication which intentionally leaves arbitrary time slots for unscheduled traffic and thus improves end-to-end delay of this type of traffic. Authors in [109] proposed a Tabu Search-based approach for generating TT transmission schedules within TTEthernet networks. However, this work, unlike [107] considered RC traffic as well as variable size time slot during the scheduling process. This work was extended in [110] in a way that it uses a Tabu search-based solution for framing data, mapping frames to virtual links and also selecting a forwarding route of each virtual link before computation of TT transmission schedules. The authors in [111] developed a hybrid genetic algorithm to generate the static schedule table of TT frames in TTEthernet networks. This work optimizes the number of allocated TT time slots and consequently improves the transmission efficiency of TTE communication.

Unlike the scheduling strategies as mentioned earlier, which only address message scheduling, Zhang et al. [112] developed a Mixed Integer Programming (MIP) multi-objective optimization algorithm to schedule TT messages and non-preemptive jobs simultaneously. Similarly, Craciunas et al. [113] introduced an SMT-based solution to co-synthesize schedules of TT communication and non-preemptive application tasks. This research, also, proposed an incremental scheduling strategy which scales to larger and more complex time-triggered systems. In [114], the same authors extend their SMT-based solvers [113] in a way that they support preemptive application tasks as well as non-preemptive ones.

All stated scheduling strategies are tailored to TTEthernet networks. However, the focus of this chapter is the scheduling problem resulting from GCL. TSN and TTEthernet standards, however, both provide real-time capabilities while sharing significant similarities, but they have some essential differences. For instance, a specific TT flow in TTEthernet comprises a single frame while a TT stream in TSN may comprise several frames. Additionally, a transmission schedule in TTEthernet networks is specified per TT flow mainly because each TT flow has its own buffer whereas in TSN TT frames of different streams are directed to TT queues, and thus schedule tables are determined for each queue. Thereby, TTEthernet specific schedulers cannot be directly used to synthesize GCLs of TSN systems [115].

For this reason, authors in [96] first determine scheduling constraints of TAS in multi-hop switched networks and then compute the valid schedule using Satisfiability Modulo Theories (SMT) and Optimisation Modulo Theories (OMT) solvers. They also based on various experiments verified that GCL offers deterministic delivery of TT messages. Pop et al. [79] designed an ILP-based solver to synthesize port-specific GCLs for each TSN capable device. This work computes the global schedule while it allocates resources to TT flows optimally. Authors in [116], in addition to standard scheduling constraints, consider the load of traffic which is directed to a specific port during synthesizing GCLs. Namely, they first select forwarding paths based on traffic load and then compute port-specific GCLs based on chosen routes. This approach leads to shorter transmission makespan compared to strategies which neglect the impact of port congestion on GCL computations. Durr et al. [117] proposed a Tabu search-based solver for scheduling of TT communication in TSN systems. This method enhances network utilization by optimizing the makespan of TT

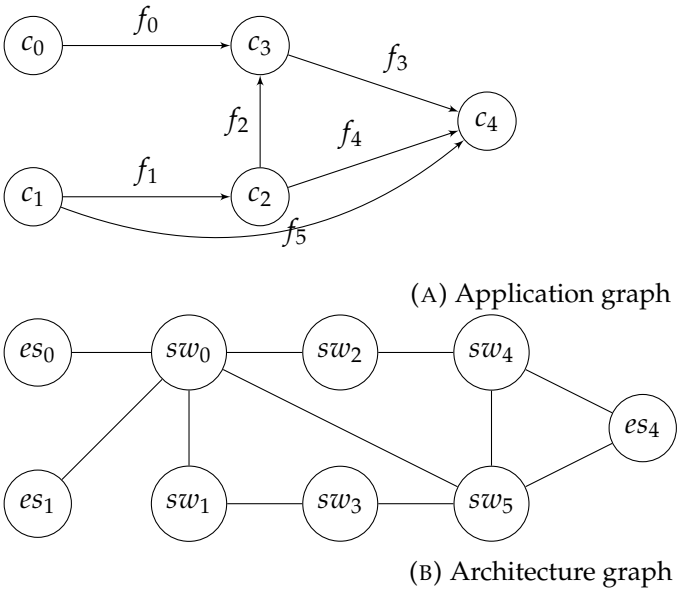
transmission schedules, but it supports neither application-specific periodicity nor job scheduling. Gavrilut et al. [118] introduced a strategy for solving the routing and scheduling problem of TT communication in TSN systems while considering the impact of AVB traffic in this context. As a result, this approach finds solutions where the timing requirements of both TT traffic and AVB streams are met. To this end, they firstly use a K-Shortest Path (KSP) algorithm [119] to compute forwarding routes of TT traffic and then generate transmission schedules using a Greedy Randomized Adaptive Search Procedure (GRASP)-based approach [120]. All solutions mentioned above, first calculate the valid routes of TT frames and then use the fixed routing information for computing the global schedule. As a result, they neglect the vital role of routing in the scheduling process.

Smirnov et al. [105] proposed a set of Pseudo-Boolean (PB) constraints to solve routing and scheduling problems of TT communication in a single-step. Furthermore, this work employs multi-objective optimization to the design space deriving from joint routing and scheduling constraints. This implementation first uses a time-consuming ILP-based approach to define the design space from PB constraints. Then, it applies the NSGA-II optimization algorithm to the exploration model. This solution does not support application-specific periods for different TT flows which is essential for TSN deployment. Instead, it is assumed that all scheduled traffic is sent in the same cycle. In addition to this study, the authors in [106] developed the ILP-based scheduling solution for the joint routing and scheduling problem and evaluate the experimental results of different traffic patterns and network topologies using two performance metrics (i.e. end-to-end delay and scheduling capability). Nayak et al. [121] also solved the routing and scheduling problem of time-triggered systems in a single step using different ILP-based solvers. This work used various optimization algorithms to ensure that the computation of transmission schedules for large networks would be completed within seconds. Authors in [122] studied how TT traffic load and the size of the network effect execution time of an ILP-based solver which employs joint routing and scheduling constraints. This research enables synthesizing of port-specific GCLs, assuming zero-queuing for TT traffic. However, it does not optimize the makespan of transmission schedules and also network utilization through different optimization techniques like load balancing. Due to the ILP-based scheduling process, the solutions above are rather time-consuming specifically for large-scale time-triggered networks. They also do not consider job scheduling and inter-flow dependencies in their experiments.

We develop a fast GA and Heuristic List Scheduler to address the interdependence of routing and scheduling constraints. These algorithms can compute the global schedule for many real-world scenarios within reasonable time intervals. Due to the combined routing and scheduling constraints, our scheduling strategies provide solutions where a scheduler that uses fixed routing would fail to find a legitimate solution. Additionally, our solution makes the distribution of real-time applications feasible using valid job bindings and resource allocations. Aside from the mentioned advantages, the GA and the HLS support job scheduling and inter-flow dependencies which to the best of our knowledge are not integrated with any TSN joint routing and scheduling solution yet.

4.2 System Model

In this chapter, the network topology and TT flows are modelled through two separate graphs: An architecture graph and an application graph. An architecture graph



(A) Application graph

(B) Architecture graph

FIGURE 4.1: An example of system model

is represented by an undirected graph G_A (R, E_l). This graph comprises TSN end-systems and switches $R = ES \bigcup SW$ as vertexes and the duplex links between them as edges. An application graph is shown by a directed acyclic graph G_C (J, E_{TT}). G_C consists of computational jobs as vertexes, and the TT flows transmitted between the jobs as edges. Since each job can have multiple successor jobs, this system modelling supports multicast TT flows as well [123]. Our scheduling algorithms use the described graphs as inputs. Figure 4.1 presents an example of the system model.

The scheduling possibilities of TT communication are derived from mapping the application graph to the architecture graph. To compute a valid system implementation, first, each computational job is assigned to a certain end-system. As a next step, the TT messages which are transmitted between a computational job and the predecessor jobs are mapped to the physical links that connect the sending end-system to one or multiple receiving end-systems.

In this model, computational jobs are intended to generate TT messages. Therefore, they can only run on end-systems, whereas TSN switches relay TT messages. It is noteworthy that the physical links $l \in E_l$ are bidirectional. Therefore, if a TT frame is traversing a specific link in one direction, simultaneously another TT message can be transmitted over the same link in reverse direction. Additionally, it is assumed in the mentioned system model that all TSN switches and end-systems are synchronized to the global time.

4.3 Problem Formulation

In TSN networks, switches and end-systems exchange three types of traffic: TT flows, AVB streams and Best Effort (BE) messages. Our scheduling algorithms are developed to generate a valid GCL so that the AVB streams and BE traffic which do not have strict timing requirements, do not interfere with the transmission of TT frames. Consequently, these algorithms (i.e. GA and HLS) only compute the static schedule table of TT messages, and non-TT frames (including AVB and BE traffic) are sent when no TT message is scheduled.

Each TT flow $e_i \in E_{\text{TT}}$ is identified by $Src_{e_i} \in J$, $Rec_{e_i} \in J$, P_{e_i} and $Size_{e_i}$. TT frames are sent periodically, therefore the P_{e_i} attribute is used to specify the periodicity of a TT flow. Since in TSN a TT flow may consist of more than one frame, $Size_{e_i}$ is equal to the number of TT frames which are sent consecutively in one P_{e_i} multiplied by each frame's length. For simplicity, it is assumed a TT flow remains in a TSN capable device just for the processing time which is computed as follows:

$$PT_{e_i} = ProcessingRate_{\text{device}} \times Size_{e_i}$$

Additionally, it is assumed that all links in an architecture graph introduce the identical propagation delay ($PropDelay_{e_i}$) to the time interval required for delivery of an arbitrary frame from a sender to a receiver. A real-time job $j \in J$ is identified by WET_{j_i} and $D_{j\text{i}}$ where WET_{ji} is the worst case execution time of the job and $D_{j\text{i}}$ determines the deadline of the job execution.

The scheduling algorithm determines the GCL of each port of a TSN capable device, and it reflects the injection time of TT flows routed via that device. The I_{e_i} determines when the sender end-system starts transmitting the TT flow just after the execution of its predecessor job (i.e. at the worst case it takes WET_{j_i} to complete). In order to offer deterministic TT communication in synchronized and scheduled networks, all port-specific GCLs begin simultaneously and repeat over the Least Common Multiple (LCM) of all P_{e_i} values called hyper-period.

4.4 Scheduling and Routing Constraints

The scheduling constraints definition of TSN systems in [96, 79] can be combined with the routing constraints as follows:

1. Each computational job is assigned to exactly one end-system. The end-system where the job can run on is chosen from the eligible end-systems $j.CanRunOn$ for that specific job. The network designers provide this information within the application graph using the knowledge of application requirements and end-system capabilities.

   $$\forall j \in J, es \in j.CanRunOn : j.processor = es$$

2. To eliminate loops, each frame of a TT flow can only pass through a certain node at most once. The $Route_{e_i}$ consists of all adjacent links which form the path from the sender to the receiver. It is noteworthy that $Route_{e_i}$ is set to one of the routing possibilities between the sender and the receiver.

   $l_i = (u, v) \in E_{\text{l}} : R = \{(l_i, .., l_{i+n}), ..., (l_j, .., l_{j+m})\}$

   $$r \in R : Route_{e_i} = r$$

3. Each TT flow can be routed through a specific link, if it can have an exclusive access to the physical link for the duration of $TransDelay_{e_{l_i}} + PropDelay_{e_{l_i}}$ just after the transmission starts. The transmission delay of a TT flow on a certain link is calculated as follows:

   $$l_i \in E_{\text{l}} : TransDelay_{e_{l_i}} = \frac{Size_{e_i}}{Bandwidth_{l_i}}$$

   Where $Bandwidth_{l_i}$ specifies the bandwidth of link l_i.

It is important to note that it is assumed a TSN capable device (including end-systems and switches) dedicates only one queue per port to TT traffic. Hence, to eliminate interleaving of different TT flows in a single TT queue, the device follows the flow isolation constraint introduced in [96]. This constraint is reflected by considering an exclusive access to the egress port and the attached link for a period of $PT_{e_i} + TransDelay_{e_{l_i}} + PropDelay_{e_{l_i}}$.

This constraint is applied to all adjacent link in $Route_{e_i}$. For each link, the time interval of exclusive access is calculated with respect to the period of $PT_{e_i} + TransDelay_{e_{l_i}} + PropDelay_{e_{l_i}}$ on previous adjacent links within $Route_{e_{l_i}}$. For minimizing the makespan of TT applications, we do not permit any gap between the hops with a duration of $PT_{e_i} + TransDelay_{e_{l_i}} + PropDelay_{e_{l_i}}$ on two subsequent links of $Route_{e_i}$. This means the buffering of TT frames is not allowed in the system model and devices follow the store and forward approach for switching TT packets.

$l_i = (u, v), l_{i+1} = (v, k), \forall (l_i, l_{i+1}) \in Route_{e_i}$:

$$I_{e_{l_{i+1}}} = I_{e_{l_i}} + PT_{e_i} + TransDelay_{e_{l_i}} + PropDelay_{e_{l_i}}$$

Based on this assumption, the injection time of a TT flow on each device can be easily calculated, and the corresponding GCL can be generated.

4. In the system model, TT flows are not restricted to one period and can be transmitted over different cycles. For this reason, the time interval of exclusive access on every link of $Route_{e_i}$ is calculated considering the periodic accesses of other TT flows which traverse the same physical links throughout their paths from the senders to the receivers.

5. Each computational job can start only when the TT flows that are sent by the predecessor jobs towards this job are delivered. In other words, the job can start transmitting TT messages only when the computing job is executed, and all predecessor TT flows are received. The flow's end to end delay determines the time interval between the injection time of a flow and its arrival time at the destination.

   $\forall e_i \in E_{\mathrm{TT}}, \forall \overline{e_i} \in pre(e_i), j \in J, Src_{e_i} = j$:

   $$e2eDelay_{\overline{e_i}} = \sum_{l \in Route_{\overline{e_i}}} PT_{\overline{e_i}} + TransDelay_{\overline{e_{l_i}}} + PropDelay_{e_{l_i}}$$

   $$I_{\overline{e_i}} + e2eDelay_{\overline{e_i}} + WET_j \leq I_{e_i}$$

   This constraint reflects inter-flow dependencies and provides an opportunity to transmit TT flows based on predefined priorities.

6. Each TT flow that is sent by a computational job must be delivered to the successor job within its deadline.

   $\forall e_i \in E_{\mathrm{TT}}, Rec_{ei} = j$:

   $$I_{e_i} + e2eDelay_{e_i} \leq D_j$$

4.5 GA implementation

The scheduling problem of time-triggered networks can be solved by combining bin-packing and a genetic algorithm [111]. We introduce a Genetic Algorithm (GA)

to generate the valid schedule with an optimized transmission time of TT messages. To be more specific, the main goal of the GA is to minimize the makespan of TT communication by optimizing the end-to-end delay as a measurement metric.

$$\min(\max_{\forall e_i \in E_{\mathrm{TT}}}(I_{e_i} + e2eDelay_{e_i}))$$

4.5.1 Individual Definition

In GA, a genome builds an individual. Each genome contains an array of genes. For resource allocation and job binding, the GA needs one gene per job. Each job-specific gene contains all end-system IDs that the job can run on (i.e. introduced in j.CanRunOn). In addition, each TT flow is mapped to one gene. The flow specific gene includes the flow's routing possibility indexes. Sets of integer numbers encode each gene.

4.5.2 Population Initialization

The GA was implemented using GAlib [124] that supports different genetic algorithms in C++. The GA first initializes an individual using information derived from the system model (including G_A and G_C). Then, it generates an initial population. In each generation, the GA chooses the individuals with the best fitness and creates a new population of individuals using the simple-point crossover operator. As a result, the best individuals are preserved for the next generation.

4.5.3 Fitness Function

The fitness function assigns a fitness score for each individual. In the GA, the fitness function first computes the global schedule of each individual and returns the makespan as a fitness value. After that, the GA evaluates the eligibility of individuals based on their fitness scores and selects the ones with the best fitness for creating the next generation.

Algorithm 1 Fitness Function

1: **procedure** FITNESS(Genome g)
2: $makespan \leftarrow 0$
3: $E_{\mathrm{TT.sorted}} \leftarrow$ *sort flows based on interdependencies*
4: $\forall e_{\in} E_{TT.sorted}$:
5: $J_s \leftarrow Src_{e_i}$
6: $J_r \leftarrow Rec_{e_i}$
7: $J_s.processor \leftarrow p \in J_s.CanRunOn$ *job's genes*
8: $J_r.processor \leftarrow p \in J_r.CanRunOn$ *job's genes*
9: $Route_{e_i} \leftarrow r \in$ *Routes using flow's genes*
10: $I_{e_i} \leftarrow$ find earliest feasible time slot
11: $Arrival_{e_i} \leftarrow I_{e_i} + e2eDelay_{e_i}$
12: **if** $Arrival_{e_i} > D_{J_r}$ **then return** infinity
13: $StartTime_{J_r} \leftarrow \max(StartTime_{J_r}, Arrival_{e_i})$
14: $FinishTime_{J_r} \leftarrow StartTime_{J_r} + WET_{J_r}$
15: $makespan \leftarrow \max(makespan, FinishTime_{J_r})$
16: **return** makespan

Algorithm 1 presents the GA's fitness function in more details. In this function, first the TT flows are sorted based on their interdependencies. For each TT flow

in $E_{\text{TT.sorted}}$, the sender and receiver jobs are assigned to the available end systems using job-specific genes in a genome. The $Route_{e_i}$ is also selected from the possible routes between sender and receiver using the flow's genes. For finding all possible routes we use the multiplication of adjacency matrix approach.

After initialization, the function using constraints (3) and (4) finds the earliest time instant that the sender job can access all adjacent links in the $Route_{e_i}$ exclusively. If the flow's injection time violates the constraint (6), it means that the individual leads to an infeasible system-wide schedule. Therefore, the function returns infinite as a fitness score to eliminate inheritance of infeasible individuals to the next generation. Line 13 corresponds to the constraint (5) and updates the start time of the receiver job accordingly. In the last line, the function returns the makespan as a fitness score. The makespan corresponds to the time instant that the execution of all computational jobs is completed. In each generation, the best solutions (individuals with minimum makespan) are stored, and the new population of individuals in the next generation is compared to the current best candidate. If the individual's makespan is bigger than the current minimum makespan, the individual will not be carried over to the next generation. Consequently, the GA converges faster to a feasible global schedule.

The GA's objective is to find the global schedule with the minimum makespan. This optimization process has the following advantages: 1) The scheduling capability will be improved since the transmission time of TT flows is optimized. 2) The optimized makespan leads to more compact transmission schedules of TT flows. Therefore, the number of guard bands that are reserved before each TT time slot to avoid interference with non-TT traffic is reduced significantly. 3) This also results in better bandwidth utilization and shorter waiting time of non-TT messages which are blocked due to exclusive TT time slots.

4.6 Heuristic List Scheduler

In addition to the GA, a Heuristic List Scheduler (HLS) is developed to generate the optimized global schedule for the transmission of TT messages. Similar to the GA, the primary goal of HLS is to minimize the makespan of TT communication by applying joint routing and scheduling constraints. Algorithm 2 presents the HLS in more details.

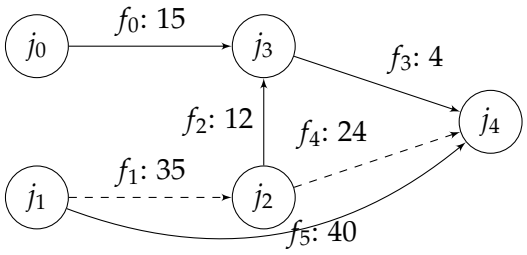
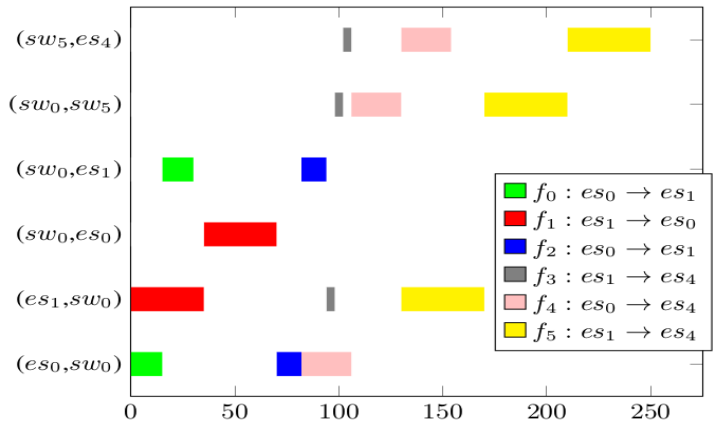


FIGURE 4.2: An application graph, weight on each edge is $Cost_{e_i}$

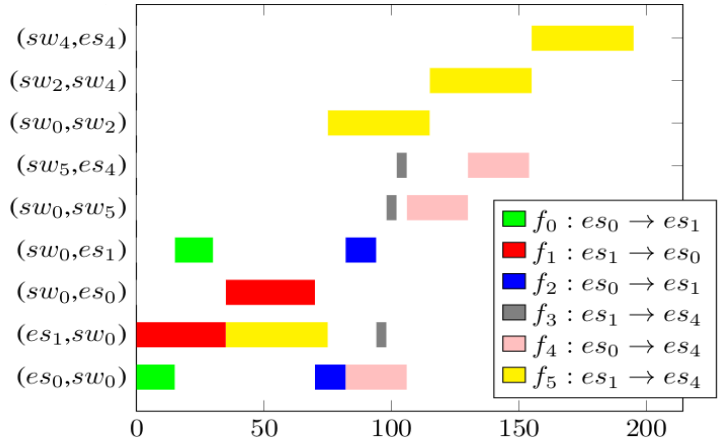
As shown in Algorithm 2, HLS first calculates the priority of each job based on the critical path. The job's critical path defines the longest path from a predecessor job to the job according to the communication cost (i.e. $Cost_{e_i} = PT_{e_i} + Size_{e_i}$). For example, in Figure 4.2 the critical path of job j_4 is shown in dashed-lines and the priority is set to 59 (i.e. $Cost_{f_1} + Cost_{f_4}$). After that, HLS sorts computational jobs based

Algorithm 2 Heuristic List Scheduler

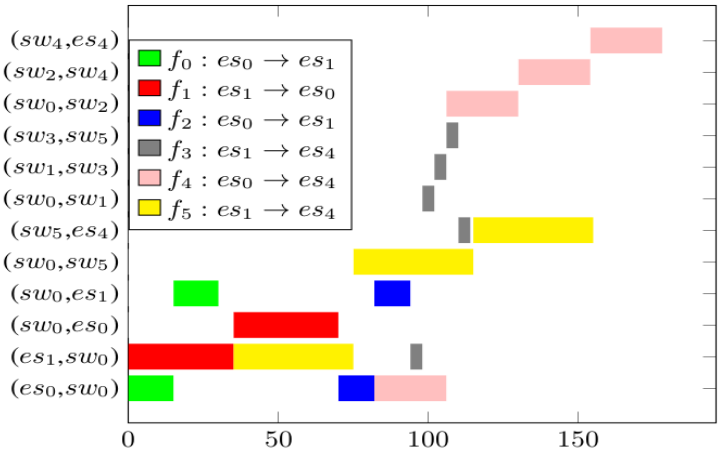
```
1: procedure LISTSCHEDULER
2: makespan ← 0
3: assign priority to each computational job
4: J_j.sorted ← sort jobs descendingly based on priorities
5: ∀j ∈ J_j.sorted is not scheduled:
6:     makespan ← Scheduler(j)
7: return makespan
8: procedure SCHEDULER(job j)
9:     if job j is not scheduled and has incoming TT flows then
10:        ∀e_i ∈ E_e_i.incoming: Scheduler(Src_e_i)
11:        is_pred_jobs_scheduled ← true
12:    else if is_pred_jobs_scheduled or job j has no child then
13:        ST ← 0
14:        for p ∈ j.CanRunOn do
15:            StartTime_j ← 0
16:            for e_i ∈ E_e_i.incoming do
17:                Arrival_e_i ← 0
18:                Arrival ← 0
19:                Routes_e_i ← FindRoutes(sender, receiver)
20:                for r ∈ Routes_e_i do
21:                    InjectTime ← FindEarliestTime
22:                    Arrival ← InjectTime + e2eDelay_e_i
23:                    if Arrival > D_j then:
24:                        go to the next end-system
25:                    if Arrival_e_i == 0 or Arrival < Arrival_e_i then
26:                        Arrival_e_i ← Arrival
27:                        Route_e_i ← r
28:                        I_e_i ← InjectTime
29:                StartTime ← max(StartTime, Arrival_ei)
30:            if StartTime_j == 0 or StartTime < StartTime_j then
31:                StartTime_j ← StartTime
32:                j.processor ← p
33:    makespan ← max(makespan, StartTime_j + WET_j)
34: return makespan
```



(A) LS TT transmission schedule



(B) HLS TT transmission schedule



(C) GA TT transmission schedule

FIGURE 4.3: Transmission schedules of LS, HLS and GA

on their priorities. In line 5, HLS schedules jobs from the highest priority to the lowest one. For each job, first, the job's incoming TT flows are retrieved. If the job needs to receive TT flows from other computational jobs before it can start transmitting TT messages (as formulated in constraint 5), HLS schedules all preceding jobs first (c.f. line 10). For instance in Figure 4.2, HLS specifies the injection time of f_0, f_1 and f_2

before it allocates a time slot to f_3. On the other hand, if the job does not have any incoming flow or all predecessor jobs are already scheduled, an available end-system from *j.CanRunOn* is allocated to the job. Then, HLS initializes *ST* and *Arrival* parameters to 0 where *ST* and *Arrival* are used to store the possible start time of the job and the possible arrival time of each incoming TT flow respectively. Further, it finds all the possible routes between the sender and receiver end systems considering constraint two and using the multiplication of adjacency matrix approach. For each routing possibility, HLS finds the earliest injection time considering constraint 3 and 4 (c.f. line 21). If the earliest injection time violates the constraint 6 then the algorithm assigns another end-system to the job and repeats the same procedure to find the next best route, which results in the optimized makespan (c.f. line 24). Constraint 6 should not be violated because it will lead to infeasible global schedules. To find the schedule with optimized makespan, the route that results in the minimum arrival time $Arrival_{e_i}$ is selected and $Route_{e_i}$ and I_{e_i} are updated accordingly (c.f line 26-28). Further, the possible start time of the job is calculated for each potential end system assigned to the job. In the end, the job is assigned to the end system that results in the minimum start time of the job and consequently the optimized transmission makespan (c.f. line 30- 32).

Aside from GA and HLS, a two-step List Scheduler (LS) is developed which solves the scheduling problem of TT communication using fixed routing. LS serves as a baseline for the two-step state-of-the-art schedulers and is used to evaluate the GA and HLS, which are joint routing and scheduling strategies in terms of scheduling capability and efficiency. The implementation of LS follows the same principles as Algorithm 2. The only difference is that LS, instead of examining all potential routes between a sender and a receiver end systems, only consider the shortest path in the process of task scheduling.

	period/ deadline (μs)	route LS	route GA
f_0	500 / 100	es_0, sw_0, es_1	es_0, sw_0, es_1
f_1	400 / 350	es_1, sw_0, es_0	es_1, sw_0, es_0
f_2	400 / 250	es_0, sw_0, es_1	es_0, sw_0, es_1
f_3	500 / 380	es_1, sw_0, sw_5, es_4	$es_1, sw_0, sw_1, sw_3, sw_5, es_4$
f_4	500 / 250	es_0, sw_0, sw_5, es_4	$es_0, sw_0, sw_2, sw_4, es_4$
f_5	1000 / 350	es_1, sw_0, sw_5, es_4	es_1, sw_0, sw_5, es_4

TABLE 4.1: TT flow parameters

To illustrate the difference between GA, HLS and LS, we use the system model in Figure 4.1. The flow's communication cost and periods are given in Figure 4.2 and Table 4.1 respectively. Table 4.1 also shows that LS always uses the shortest paths, while GA and HLS find the routing that leads to a more optimal makespan. The Gantt charts in Figure 4.3 present the global schedules that were computed by LS, GA and HLS. In the Gantt charts, each box presents the time slot that is dedicated to a certain TT flow on a specific link. As the graphs show, in GA and HLS, the makespan of TT flows is improved compared to LS (from 250 μs to 178 μs and 195 μs respectively). The reason is that the fixed routes used in LS cause high traffic load on certain physical links, although other links are under low utilization. In contrast, GA and HLS benefit from the load balancing while computing routes using joint scheduling and routing constraints. The enhancement of the makespan is important

because the TT transmission schedule plays a key role in the complex time-triggered systems that comprise several mission-critical applications with short deadlines.

4.7 EXPERIMENTS AND EVALUATION

4.7.1 Experimental Setup

GA, HLS and LS are implemented in C++ and run on a T460 ThinkPad computer with 2.4GHz Intel i5 CPU and 32GB of memory.

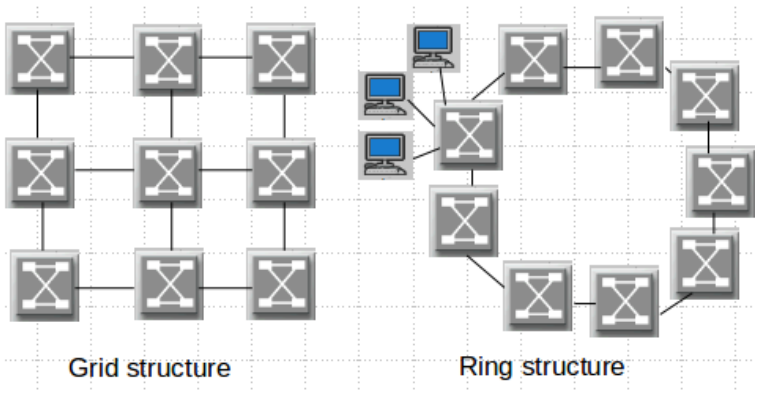


FIGURE 4.4: Topologies used in our experiments. Every switch is connected to either 3, 4 or 5 end systems in a star structure.

We generate 120 system models (including architecture and application graphs) using the SNAP library [125]. These system models are given as inputs to GA, HLS and LS schedulers. We conduct our experiments on two different network topologies: 1) meshed grid and 2) ring. As depicted in Figure 4.4, each network comprises nine switches, and every switch is attached to 3 to 5 end systems. We consider the ring topology to reflect a typical structure of industrial control networks. To evaluate our joint routing and scheduling constraints, we use the meshed grid structure. This topology has higher connectivity and provides more routing possibilities for every TT message. It is also assumed that all physical links in our experimental networks have a bandwidth of 1 Gbps and all switches require two nanoseconds for processing each byte.

The flow interdependencies are formulated using a random Forest Fire directed graph [125]. In each synthetic application graph which includes 15 jobs, we use four different traffic classes. The characteristics of each traffic class are detailed in Table 4.2. It is noteworthy that the list of eligible end-systems which every job can run on (i.e. $j.CanRunOn$) is chosen randomly. Furthermore, each TT flow's deadline is selected randomly from a range of 200 to 800 microseconds. It is also assumed that all end systems are identical, and all computational jobs have the same execution time (i.e. 4 microseconds).

The GA initializes the genetic algorithm parameters as follows: population size = 100, the number of generations = 100, the mutation probability = 0.2, the crossover probability = 0.9 and convergence probability = 1.

4.7.2 Experiments and Evaluation

In this section, the scheduling capability and efficiency of GA and HLS are evaluated using the experimental results of LS, which serve as a baseline for the state-of-art

traffic class	$Size_{e_i}$ (bytes)	P_{e_i} (μs)
$class_1$	200	100
$class_2$	400	200
$class_3$	600	300
$class_4$	800	400

TABLE 4.2: Traffic class parameters

two-phase scheduling solutions.

The first part of the experiments is intended to study the impact of varying load on schedulability and transmission makespan of LS, HLS and GA schedulers. For this purpose, the system models with the meshed grid structure (Figure 4.4) and three different TT traffic loads (i.e. 30, 35 and 40 TT flows) are considered. For every traffic load, thirty different flow interdependency patterns are generated although the network topology remains the same during this set of experiments. In short, ninety synthetic system models are used in this section of experiments.

TT flows	LS Avg makespan (μs)	HLS Avg makespan (μs)	GA Avg makespan (μs)	ImpRatio HLS	ImpRatio GA
30	168.62	60.26	83.8	0.64	0.4
35	179.42	127.77	121.14	0.28	0.32
40	180	133.51	142	0.25	0.21

TABLE 4.3: Transmission makespans for the meshed grid topology and varying load.

Table 4.3 lists the required time for delivering all TT flows to corresponding destinations. The improved ratio of makespan in this table is calculated as follows:

$$ImpRatio = \frac{LS\ average\ makespan - HLS/GA\ average\ makespan}{LS\ average\ makespan}$$

Both HLS and GA compared to LS improve TT transmission efficiency on average by 0.39 and 0.31, respectively. The enhanced scheduling efficiency of both joint routing and scheduling strategies (i.e. HLS and GA) imply shorter end-to-end delays, more compact TT global schedules and better utilization of link bandwidth. To achieve the optimal makespan, the mentioned strategies distribute the computational jobs over available end-systems and balance the load over different physical links. Additionally, they schedule different TT flows on a certain link with the minimum gap between their time slots. Furthermore, joining time slots of consecutive TT flows on a specific link leads to the minimum number of guard bands and better resource utilization (e.g. link bandwidth).

Table 4.4 presents the average execution time of HLS, GA and LS for each traffic load. As simulation results show, LS solves the scheduling problem faster than GA and HLS since LS uses the fixed routing and ignores the crucial role of routing in the scheduling process. In other words, the fixed routing between each sender and receiver limits the search space and reduces the solving time of LS significantly. In contrast, GA considers all routing possibilities and employs joint routing and

TT flows	LS Avg Exec Time (s)	HLS Avg Exec Time (s)	GA Avg Exec Time (s)
30	0.01	0.14	52.54
35	0.013	0.28	54.61
40	0.014	0.33	56.75

TABLE 4.4: Execution time for the meshed grid topology and varying load.

scheduling constraints. Therefore, the design space of system implementations gets bigger, and both approaches (i.e. HLS and GA) require more time for the global schedule generation.

Besides, as results depict, the average execution time of HLS, GA and LS increase when the number of flows increases mainly because, by increasing network load, all described schedulers take more time to find a job binding and resource allocation that leads to a valid schedule.

It has to be noted that HLS computes a valid schedule in a shorter time compared to GA, mainly because the GA-based algorithm is very time-consuming.

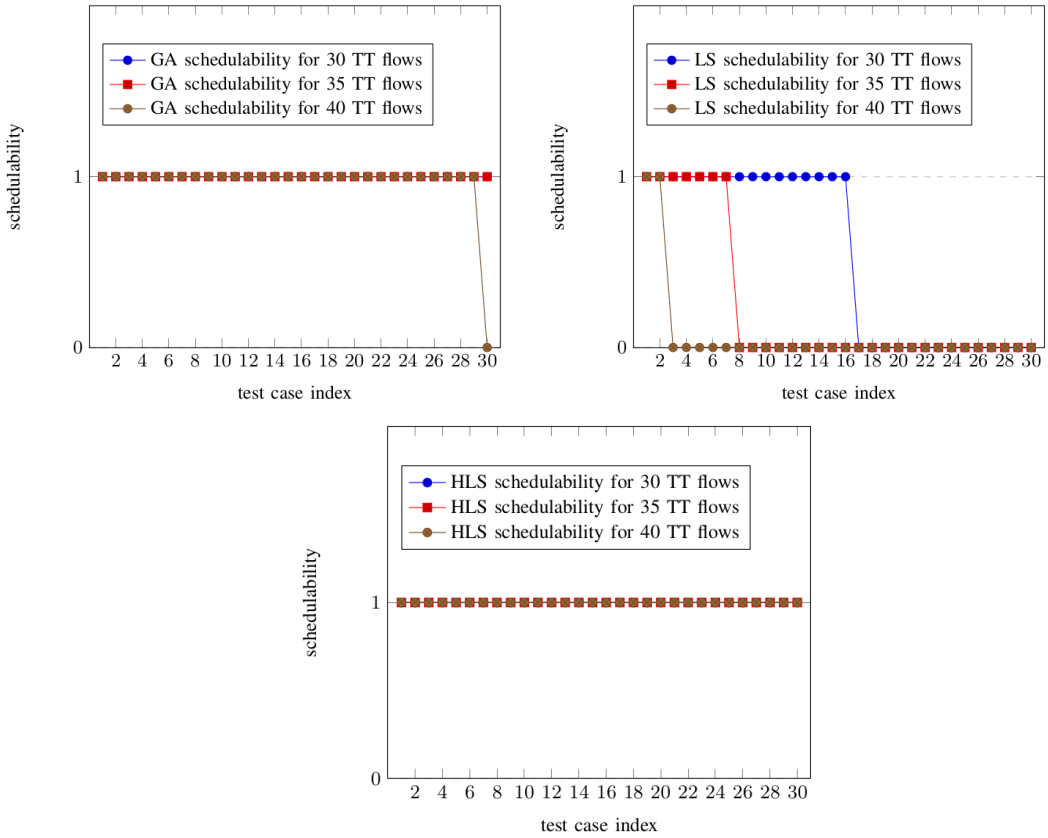


FIGURE 4.5: Schedulability of GA, LS and HLS with varying TT loads and the meshed grid topology

The graphs in Figure 4.5 depict the schedulability of GA, LS and HLS for the above benchmarks. The schedulability ratio of GA and HLS for test cases with varying loads is on average, 0.98 and 1, respectively. This implies that HLS computes the

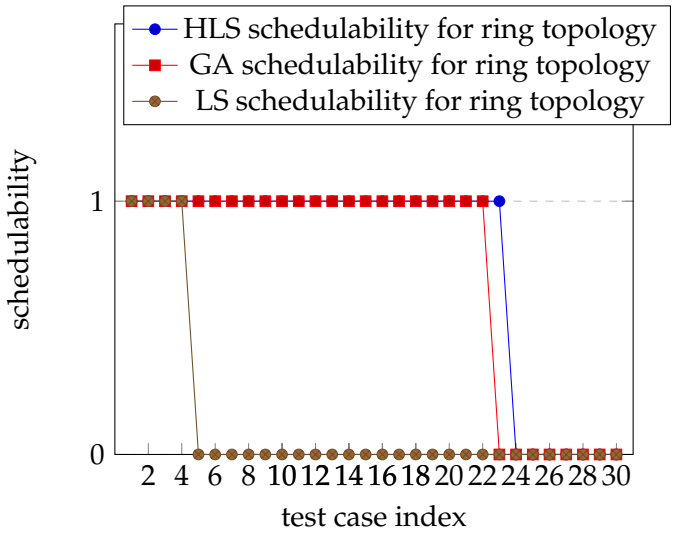


FIGURE 4.6: Schedulability of HLS, GA and LS with different topologies

global schedule for all test cases in this set of experiments while GA fails to find a solution for one of the above benchmarks. On the contrary, LS scheduling ratio for the same test cases is on average, 0.27. As the graphs illustrate, the scheduling capability of LS compared to GA and HLS decreases significantly when the network utilization (i.e. the number of TT messages in our experiments) increases. LS, like other state-of-the-art scheduling solutions, solves the scheduling and routing problems separately. Thereby, the increasing number of TT flows in the LS scheduler may lead to over-utilized links and the violation of constraint 6. GA and HLS resolve this issue by transmitting TT flows over different routes and avoiding bottlenecks in routing TT frames. In other words, GA and HLS both excel LS by examining different routing possibilities during the scheduling and optimization process.

However, in the mentioned experiments, GA and HLS, both outperform LS regarding TT transmission efficiency and capability, but HLS has a higher schedulability ratio compared to GA.

To evaluate the effects of the network topology on GA and HLS performance metrics, we repeat the test cases with 30 flows in the first part using the ring structure (Figure 4.4). To be more specific, the flow interdependency patterns in this set of benchmarks stay unchanged although the switches are connected in a ring topology rather than a meshed grid structure. For this part of the experiments, GA changes the value of the number of generations to 800. However, the other parameters of the genetic algorithm remain intact.

According to experimental results which are presented in Figure 4.6, when the ring topology is used, the LS capability to find a valid schedule declines significantly compared to the HLS and GA scheduler. It is essential to note the similar number of flows in the ring structure leads to a higher network utilization. Therefore, LS fails to meet timing requirements of more test cases due to the violation of constraint 6. In contrast, HLS and GA offer a higher schedulability ratio than two-phase LS for the test scenarios with a ring network topology since HLS and GA overcome the limitation of LS regarding over-utilized links by balancing loads over different routes. To this end, HLS and GA examine a higher number of scheduling possibilities that are derived from the joint scheduling and routing constraints. Additionally, HLS and GA require, on average 0.06 and 182.7 seconds respectively to solve the joint

routing and scheduling problems of this set of system models. On the other hand, LS computes the static schedules for these test cases on average within six ms. It is noteworthy in this part of experiments that GA takes more time to converge to the solution since it is initialized to a higher number of generations compared to the previous experimental section.

Chapter 5

Fault-Tolerant Scheduler for Time-Triggered Communication in Time-Sensitive Systems

The goal of the chapter is to present a time-triggered scheduler for TSN-based applications which optimizes the overall system reliability based on application and platform models while satisfying the real-time constraints of the application. The system reliability considers the (1) redundancy in the application models (e.g. redundant and non-redundant real-time jobs), (2) the redundancy in the platform models and the reliability of the TSN platform components (e.g. end systems, switches and links) and (3) novel TSN-based fault-tolerance mechanisms such as FRER.

This chapter also introduces a new reliability analysis technique for safety-critical systems. The reliability model comprises real-time jobs as system constituents and interactions between jobs as control dependencies. This technique analyzes the reliability of a TSN system as a function of the reliability of its real-time jobs and the control transfers between them in forms of TT messages. This approach uses the reliability of TT messages that are exchanged between different real-time jobs as an input for computing the system reliability. For calculating the reliability of message transmissions, first, it is essential to select the redundant paths for the message. To achieve that, it is assumed at least two forwarding paths exist for each TT message in the network. Then, using the reliability of the platform components which participate in the message transmission and also based on the principles of reliability of series and parallel system [126], the reliability of message transmissions is calculated. This reliability analysis technique determines the impact of different safety-critical jobs and network components on the overall system reliability. Therefore, the system designers can choose network components and additionally plan the TSN network more efficiently.

In addition to the precedence constraints between predecessor and successor jobs, this work supports conditional precedence constraints between different real-time jobs. A conditional precedence constraint specifies whether the incoming TT message from a predecessor job is essential to execute the job or is substitutable with the TT messages from other predecessor jobs for commencing the execution of the job.

The computation of a TT communication schedule is NP-complete. Besides, the distribution of real-time applications and the ever-increasing number of network components leads to a more compute-intensive TT scheduling process. Consequently, several works focus on reducing the computational complexity of the TT scheduling problem by making different abstractions. Most of the state-of-the-art

TT schedulers [96, 79, 111, 116, 118] solve the routing and scheduling problems sequentially. A few recent works [105, 106, 121, 122] proposed ILP-based solutions that employ joint routing and scheduling constraints. These solutions are very time-consuming and not scalable to large real-time systems. Besides, they do not consider inter-flow dependencies and job scheduling. Therefore, in Chapter 4 a Genetic Algorithm (GA) was presented that addresses the impact of the routing problem on the scheduling constraints while supporting application-specific periods, precedence constraints and job scheduling.

Aside from the aforementioned simplifications, the majority of TT schedulers [96, 79, 111, 105, 106, 116, 121, 122] assume that the communication infrastructure is fault-free. However, in practice, the network can experience changes such as solicited ones like reconfigurations or unsolicited ones like failures while exchanging messages. In TSN networks, due to stringent temporal and safety requirements of real-time systems, faulty behaviours must be mitigated using redundancy. The redundancy can be temporal or spatial [127]. For temporal redundancy, more than one copy of the message is scheduled to be sent during one period. This type of redundancy protects TSN networks against transient and intermittent failures, although the temporal redundancy can not handle the permanent faults. To alleviate permanent failures, the TSN task group develops a new spatial redundancy mechanism that is called Frame Replication and Elimination for Reliability (FRER) [28]. In FRER, every message is replicated and transmitted over one or more redundant paths. Thus, FRER offers bounded end-to-end latency and low packet loss which are the major concerns of mission-critical systems. The seamless recovery from faulty behaviours is vital for cyber-physical systems, since failures in such systems may result in irreparable environmental damages and huge financial losses.

To support FRER features and also benefit from fault-tolerant communication, the message replication, elimination of replicas and the redundant path selection need to be considered in the process of the message scheduling. These considerations lead to a bigger search space for valid transmission schedules and also a more complicated schedule exploration process. Hence, optimization algorithms that facilitate finding the schedule for fault-tolerant TT communication from the vast system design space play an essential role in the deployment of TSN networks. In this chapter, the GA [22] and HLS [21] which are introduced in Chapter 4 and employ the joint routing and scheduling constraints, are extended in order to meet the reliability requirements of real-time systems. This work mainly focuses on permanent hardware failures, although the FRER redundancy mechanism also alleviates transient failures. For this purpose, it is assumed that each message is duplicated at designated devices and each copy is sent over only one disjoint path mainly because the scheduler needs to allocate additional network resources like bandwidth for each copy of the message and an increase in the number of message replicas increases network utilization considerably. Additionally, the permanent crash failures (e.g. link failure) are not as frequent as transient failures [126]. Therefore, this scheduling scheme which considers only duplication of TT messages and forwarding them over the redundant route offers a reasonable degree of reliability.

It has to be noted that the fault-tolerant scheduler proposed in this chapter is the first scheduling solution that is developed based on TSN redundancy management (i.e. FRER). To this end, the fault-tolerant TSN scheduler takes into consideration the message replication, redundant path selection, and message elimination during the TT scheduling process. Therefore, this solution can play a vital role in the deployment of TSN. Apart from the aforementioned distinctive features, this work is

not limited to any specific network topology and applicable to a wide range of time-triggered systems.

The remainder of the chapter is structured as follows: Section 5.1 discusses related work on fault-tolerant TT scheduling. Section 5.2 describes the system model used in this work. In Section 5.3, the scheduling problem of fault-tolerant TT communication is formulated. In Section 5.4, the reliability analysis technique is described. The following sections describe the fault-tolerant GA and HLS. In the last section, experimental results are evaluated.

5.1 Related Work

In the last years, several studies have been done on the scheduling problem which arises from the TAS concept. In [96, 79, 111, 116, 118], scheduling solutions use fixed routing to synthesize the GCL of TSN-aware devices. These works ignore the inter-dependencies of routing and scheduling constraints, and consequently, they provide sub-optimal solutions. To address this issue in Chapter 4, GA-based and list scheduling schemes are developed that apply the routing and scheduling constraints in a single-step. These schedulers, unlike the few recent works [105, 106, 121, 122] that also follow similar principles (i.e. joint routing and scheduling constraints), support inter-flow dependencies and job scheduling. These features play key roles in the deployment of TSN networks since they are essential for modern cyber-physical systems.

However, all discussed scheduling solutions optimize the length of the transmission schedule while fulfilling the temporal requirements, but they assume that the network is fault-free during the execution of safety-critical applications and does not face any failures. This assumption is very optimistic, and in practice, the system constituents encounter different types of failures over time. The complexity of the TT scheduling problem in fault-free real-time systems is already quite high. Hence, fulfilling the fault-tolerance requirements of the safety-critical system leads to an even more computationally complex scheduling process. To simplify this process, several works investigate different aspects of the fault-tolerant scheduling problem. For instance, in [128, 129, 130, 131] different scheduling mechanisms were introduced for real-time systems with multiprocessor architectures. These solutions addressed potential processor failures using a Primary-Backup (P/B) approach. Qin et al. [132] developed a scheduler that can only mitigate one processor failure. This work computes the primary schedule using a greedy scheme and deploys the backup schedule in case of fault occurrence. Authors in [133] proposed a Contention-Aware Fault-Tolerant (CAFT) scheduler which addresses the occurrence of an arbitrary number of processor failures using active replication. This work did not require any further fault detection and recovery strategy. Likewise, Izosimove et al. [134] introduced several optimization techniques which combine active replication and re-execution of jobs in order to compute fault-tolerant schedules. These algorithms use optimal task binding and resource allocation to prevent the need for any additional hardware resources. In [135], authors enhanced the efficiency of a replication scheme by identifying the worst-case finish time of jobs under the worst-case fault occurrences.

All studies above only consider crash failures of processors which can lead to a failure in the execution of a safety-critical application. However, fault tolerance concerning failures in message delivery is not covered in these scheduling strategies.

Avni et al. [127] proposed a method that aims to find a (k, l)-resistant transmission schedule using a CEGAR-based approach. To achieve this goal, switches

ensure that at least l copies of a particular message are delivered to the receiver application in the presence of at most k faulty links. In [136], a new schedule for TT communication was introduced that masks multiple link failures using the localized fault-tolerant protocol instead of spatial redundancy. Thereby, this scheduling scheme increases the available bandwidth for non-critical traffic, mainly when links are fault-free. Atallah et. al [137] developed an incremental algorithm for planning of an 802.1Qbv-aware network considering joint routing and scheduling constraints. This approach aimed to optimize the overall cost of the network. Unlike the first category of scheduling solutions, the aforementioned schedulers mask crash failures of links occurring during message transmission, while they neglect processor failures over the execution period.

The authors in [138, 139] developed a scheduling scheme that considered both processor and communication infrastructure failures during the scheduling process of multiprocessor computing systems. Dogan et al. [139] proposed a scheduling solution that optimizes the execution time and reliability of distributed systems simultaneously. Smirnov et al. [140] formulated the constraints for finding redundant routings in a vehicular networks in a way that the overall system reliability can be optimized. The authors showed that the state-of-art design space explorations such as SAT-Decoding are not scalable to optimization problem of the system reliability in the automotive domain. They therefore proposed an Evolutionary Algorithm (EA) where every genome is encoded efficiently considering optimization objectives (i.e. the number of critical links and Mean Time To Failure (MTTF) of communication flow). However, these works did not consider timing constraints of real-time systems and did not use redundancy to mitigate failures.

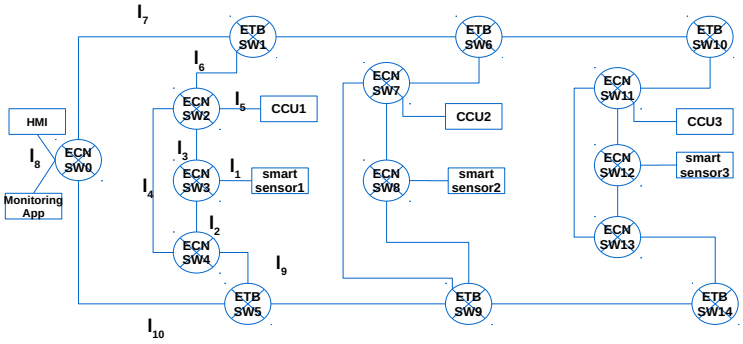
In this chapter, fault-tolerant TT schedulers are developed that meet temporal requirements of hard real-time systems while optimizing the reliability of the system. Namely, the main goal of these schedulers is to maximize system reliability. To calculate the system reliability, a novel reliability analysis technique is introduced. This technique, in contrast to state-of-the-art algorithms which aimed to mitigate either processor crashes or link failures, calculates the reliability of a system based on the reliability of every network component that has a role in the message delivery. Moreover, the conditional precedence constraints between safety-critical jobs are modelled using a conditional application graph. This conditional application graph accurately reflects the job dependencies, which leads to more realistic and efficient transmission schedules.

5.2 System Model

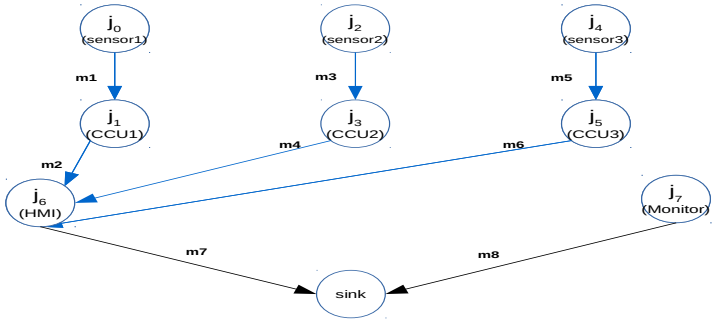
To model real-time applications and the architecture of the physical platform on which jobs run and communicate, two different graphs are defined: A conditional application graph and an architecture graph. These graphs contain all parameters of the application graph and the architecture graph that are specified in Chapter 4. However, they include new parameters to address reliability requirements.

5.2.1 Conditional Application Graph

In this work, the conditional application graph is a directed acyclic graph G_C defined by the tuple $< J, E_{\mathrm{TT}} >$. Each vertex in this graph presents one real-time job. Additionally, a dummy sink vertex j_s is also introduced in G_C to formulate the reliability of a safety-critical system. E_{TT} is a set of conditional directed edges which represent TT message transmissions between real-time jobs. A real-time job $j_i \in J$ is identified



(A) Architecture graph



(B) Conditional application graph

FIGURE 5.1: Example of system model based on train communication network

by $< WET_{j_i}, Rel_{j_i}, D_{j_i} >$ where WET_{j_i} is the worst-case execution time of the job, Rel_{j_i} defines the reliability of the job and D_{j_i} determines the deadline of the job execution. The reliability of the real-time job Rel_{j_i} is defined as the probability of correct execution of the job. The job deadline denotes the maximum permissible time by which all essential predecessor jobs provide the TT messages, and the execution of the job is finished.

On the other hand, each edge $e_i \in E_{\mathrm{TT}}$ models the TT message transmissions between two jobs. In this work, the conditional edge is represented by $< S_{e_i}, Rec_{e_i}, P_{e_i}, T_{e_i}, Rel_{e_i}, C_{e_i}, I_{e_i} >$ tuple where S_{e_i} and Rec_{e_i} specify the sender job and the receiver job of TT messages respectively, P_{e_i} defines the periodicity of the TT messages, T_{e_i} is the transmission time of TT messages, Rel_{e_i} determines the probability of successful delivery of messages to the receiver job, C_{e_i} denotes whether TT messages from the predecessor job $S_{e\mathrm{i}}$ are essential ($C_{e_i} = 1$) or substitutable ($C_{e_i} = 0$) with messages from other predecessor jobs and I_{e_i} is the injection time of the TT messages. To be specific, I_{e_i} identifies when the sender job S_{e_i} starts the transmission of the TT messages to the receiver job Rec_{e_i} after its execution. The control transfer C_{e_i} from one job to another job can be essential, meaning that the job cannot run before receiving the TT messages from the predecessor job. On the contrary, the TT messages from the predecessor job can be substituted with the TT messages from other predecessor jobs. This implies that the jobs that send the substitutable TT messages are redundant jobs. It is assumed a job may have at least two incoming conditional edges with the latter case condition C_{e_i}. Therefore, the job needs to receive an input from at least one of its predecessor jobs with the substitutable condition in order to proceed with the execution. Namely, if the job has multiple incoming TT flows with the substitutable condition, at least one of its predecessor jobs needs to deliver TT messages

before commencing the job's execution.

5.2.2 Architecture Graph

An architecture graph is an undirected graph G_A, which is presented by the tuple $< R, E_l >$. In this graph, each vertex represents either a TSN end system or a TSN switch ($R = (SW \bigcup ES)$). On the other hand, E_l denotes the duplex physical links between TSN-aware devices (i.e. TSN end system and TSN switch). From the TT scheduling perspective, the duplex link implies that a TT message can traverse a certain link while another message is sent over the same link but in the opposite direction. For each network component $c \in (R \bigcup E_l)$, the reliability Rel_{c_i} is defined as the probability of correct operation.

Figure 5.1 depicts an example of a system model. This system model is based on an example layout of a train network [141]. In this model, the sensors that reside in different consist networks sample data and forward samples to the consist's Central Computing Unit (CCU). After that, the sensor samples are forwarded to Human Machine Interface (HMI) for further processing.

5.2.3 Fault Model

In this work, it is assumed that every network component (including end systems and switches) forms a Fault-Containment Region (FCR). Thereby, the failure of each network component is considered to be independent. It is also assumed that only a single permanent hardware failure can occur in any network component which forms a separate forwarding route and engages in the message transmission. This means that if a network component fails, it will not be operational for the rest of the execution of the safety-critical applications. This solution only considers the duplication of a TT message at a device residing at one border of two disjoint routes, forwarding each copy of messages over a disjoint path and eventually the elimination of duplicated messages at a device residing at another border of two disjoint routes. However, this scheduler can be extended to tolerate several permanent component crash failures by considering more than two message replicas and redundant routes at the expense of higher network load. Furthermore, the concept of redundant real-time jobs is introduced using conditional precedence constraints. Based on this concept, if one of the redundant real-time jobs executes correctly and the rest of the redundant jobs fail, the mission-critical system can still operate correctly.

5.3 Problem Formulation

In this chapter, the fault-tolerant scheduler is proposed, which employs joint scheduling and routing constraints. These constraints are derived from the scheduling constraints defined in [96, 79] and the routing constraints as follows:

1. **Resource Allocation Constraint:** Each job is executed on exactly one end-system. The target end-system is selected from the eligible end-systems that are specified in $j.CanRunOn$.

2. **Path-Dependent Constraint:** Each TT message traverses a certain network component at most once in order to prevent loops.

3. **Contention-free Constraint:** Each TT message can be forwarded through a certain link, only if it can access the physical link exclusively for the duration of T_{e_i}.

4. **Message-Specific Periodicity Constraint:** In this system model, TT messages can be sent over different cycles. Hence, the duration of exclusive access on every physical link needs to take into account the periodic accesses of other TT messages that are sent over the same links.

5. **Inter-Flow Dependency Constraint:** Each job can run only after reception of all essential TT messages from the predecessor jobs.

6. **Delivery Deadline Constraint:** Each TT message must reach the destination within the deadline of the receiver job. The message's end-to-end delay ED_{e_i} specifies the time interval between the injection time of message I_{e_i} and its arrival time at the receiver end-system [21, 22].

   $\forall e \in E_{\mathrm{TT}}, Rec_{e_i} = j$

   $$I_{e_i} + ED_{e_i} < D_{j_i}$$

In addition to imposing the joint scheduling and routing constraints, the scheduling strategy maximizes the reliability of a safety-critical system. To achieve this goal, the reliability of a system is modelled and computed using a novel reliability analysis technique.

The fault-tolerant scheduling algorithm uses the described graphs for job binding and resource allocation. Namely, a specific end system is allocated for each real-time job, since it is assumed only TSN end systems can initiate sending of TT frames. In addition to job scheduling, the message scheduling is performed by the temporal and spatial allocation of network components ($c \in (R \bigcup E_l)$) that lie between sender and receiver end systems.

This work only focuses on the scheduling of TT communication. However, TSN-capable devices transmit different types of traffic (e.g. AVB streams and best-effort traffic). The TSN-capable devices guarantee deterministic delivery of TT messages through TAS. For this purpose, TAS does not permit non-TT streams (e.g. BE frames) to be sent during the transmission of TT messages over the same physical links. Furthermore, a fault-tolerant start-up algorithm is assumed for the real-time system [17].

5.4 Reliability Model of Safety-critical Systems

The reliability of a safety-critical system is modelled using the reliability of real-time jobs and the reliability of TT communication between them. As the first step in the reliability analysis technique, it is required to calculate the reliability of TT message delivery.

5.4.1 Reliability of Message Transmission

To deliver a TT message successfully all network components which form the forwarding paths, need to function correctly. In this work, the reliability of a network component Rel_{c_i} which has a constant failure rate, is formulated by a Poisson process as follows [126]:

$$Rel_{c_i}(t) = e^{-\lambda t} \tag{5.1}$$

The failure rate λ specifies the number of faults that a component experiences per unit of time (e.g. second). However, formulating the Rel_{c_i} using a Poisson process does not correspond to the actual failure model of a network component but

according to experimental results in [142] it leads to a practical failure modelling [139].

As stated before, every TT message is duplicated at a device which resides at a border of two disjoint paths. Then, each copy is sent over one of the routes. The device that resides at another border of these two paths forwards only one copy of the message and discards the second copy. This approach may result in high variance of the delivery latency, which can happen due to redundant paths with different lengths and also considering zero queuing for TT frames in devices. For this purpose, it is assumed that the second boundary device forwards the frame based on the arrival time of the message from the longer path. As a result, TT frames are buffered in the boundary devices until the expected arrival time of the messages from the longer forwarding route, which is different from the zero queuing assumption applied to other devices.

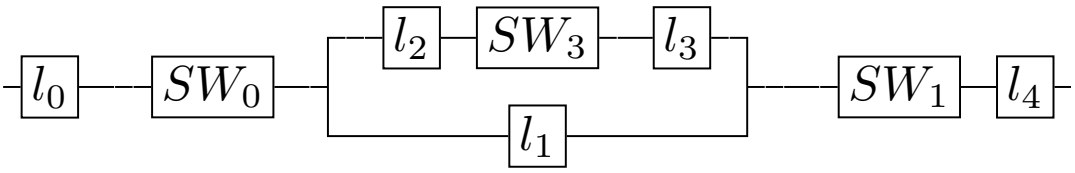


FIGURE 5.2: Reliability model of message m_1

To calculate the message reliability, the redundant path for a specific message is modelled as series and parallel systems where each network component $c \in (R \bigcup E_l)$ is represented as one module and these modules are connected based on the topology of the redundant route. For instance, Figure 5.2 illustrates the reliability model of message m_1. As it is shown in Figure 5.2, each network component is mapped to a separate module. The module has a series connection to other modules if the failure of the component causes the message transmission to fail. The reliability of a series system is calculated as follows [126]:

$$Rel(t) = \prod_{i=1}^{N} Rel_{c_i}(t) \tag{5.2}$$

Where $Rel_{c_i}(t)$ is the reliability of a network component that is part of a series system.

In contrast, if the message is delivered to the destination while at least one network component out of sets of components operates correctly, the components form a parallel system. The reliability of a parallel system is computed as follows [126]:

$$Rel(t) = 1 - \prod_{i=1}^{N} (1 - Rel_{c_i}(t)) \tag{5.3}$$

To simplify the notation of the reliability, in the rest of this chapter the reliability is not shown as a function of the time, although all reliabilities are still dependent on time. In Figure 5.1, it is assumed that all physical links are identical and thus have the same failure rate. The same assumption is made for TSN switches, meaning that all switches are identical. Therefore, the reliability of a link and a switch are denoted with Rel_l and Rel_{sw} respectively. Hence, according to the described method, the reliability of message m_1 in Figure 5.1 is equal to:

$$Rel_{m_1} = Rel_{l_1} Rel_{sw_3} [1 - (1 - Rel_{l_3})(1 - Rel_{l_2} Rel_{sw_4} Rel_{l_4})] Rel_{sw_2} Rel_{l_5}$$

and we can summarize it as follow:

$$Rel_{\mathrm{m1}} = Rel_l^2 Rel_{\mathrm{sw}}{}^2 (Rel_l + Rel_l^2 Rel_{\mathrm{sw}} - Rel_l^3 R_{\mathrm{sw}}) \tag{5.4}$$

5.4.2 Reliability of Safety-Critical Jobs

After calculating the reliability of message transmissions, it is time to compute the reliability of a safety-critical job Rel_{e_i}, which is the probability of correct execution of the job. The job correctly runs when it receives all essential TT messages from the predecessor jobs, and the dedicated end system for execution operates correctly. Therefore, the reliability of jobs which do not have any incoming TT flows is equal to the reliability of the assigned end system. In the above section, the concept of conditional precedence constrains between two jobs is introduced. The job reliability is formulated so that it reflects the C_{e_i} attribute. To calculate the reliability of a job, the principles of series and parallel systems are used. In this context, every predecessor job and corresponding TT flow are mapped to separate modules, and the topology of a system is defined based on the conditional control dependencies.

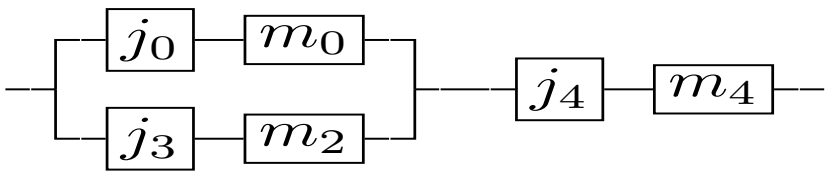


FIGURE 5.3: Reliability model of job j_6

Figure 5.3 presents the reliability model of job j_6 from Figure 5.1. As Figure 5.3 demonstrates, j_6 can start its execution only after receiving the TT frames from job j_1, however, the TT messages from either j_3 or j_5 are sufficient to proceed with the execution. Hence, the reliability of job j_6 is formulated as follows:

$$Rel_{j_6} = Rel_{j_1} Rel_{m_2} [1 - (1 - Rel_{j_3} Rel_{m_5})(1 - Rel_{j_5} Rel_{m_8})]$$

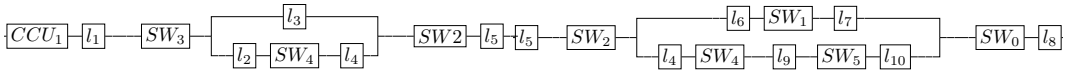
Where Rel_{j_1}, Rel_{j_3} and Rel_{j_1} are calculated as follows:

$$Rel_{j_1} = Rel_{CCU_1} Rel j_0 Rel_{m_1}, Rel_{j_3} = Rel_{CCU_2} Rel_{j_2} Rel_{m_4}, Rel_{j_5} = Rel_{CCU_3} Rel_{j_4} Rel_{m_7}$$

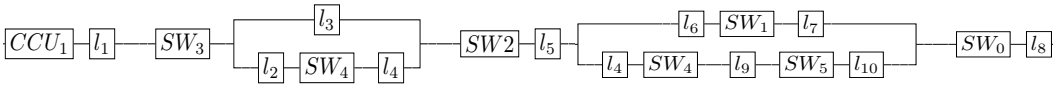
As shown in Figure 5.1, jobs j_0, j_2 and j_4 do not have any incoming edge and they run on $sensor_1$, $sensor_2$ and $sensor_3$ respectively. In this example, it is assumed that the sensors are fault free and all CCUs have the same failure rates. Thus the reliability of job j_6 can be written as follows:

$$\begin{aligned} Rel_{\mathrm{j6}} = {} & Rel_{\mathrm{CCU}} Rel_{m_1} Rel_{m_2} (Rel_{\mathrm{CCU}} Rel_{m_4} Rel_{m_5} + Rel_{\mathrm{CCU}} Rel_{m_7} Rel_{m_8} - \\ & Rel_{\mathrm{CCU}}{}^2 Rel_{m_4} Rel_{m_5} Rel_{m_7} Rel_{m_8}) \end{aligned} \tag{5.5}$$

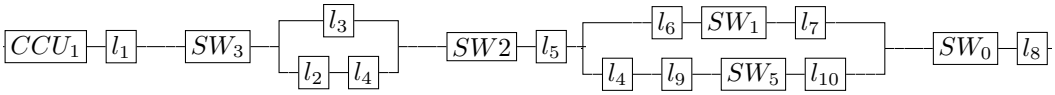
Each of the message reliabilities (e.g. R_{m1}) in the above formula needs to be calculated separately using the approach that is introduced in the previous section.



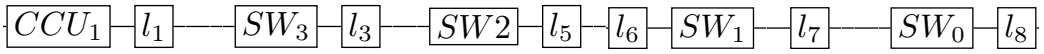
(A) Reliability model of s_1



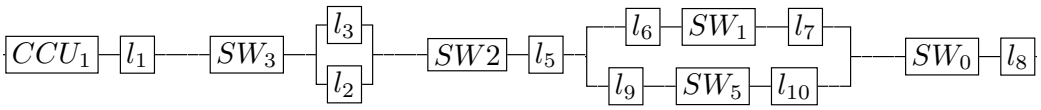
(B) Reliability model of s_1 after removing repetitive components l_5,SW_2



(C) Reliability model of s_1 after expanding on SW_4 (when SW_4 is fault-free)



(D) Reliability model of s_1 after expanding on SW_4 (when SW_4 is faulty)



(E) Reliability model of s_1 after expanding on l_4 (when l_4 and SW_4 fault-free)

FIGURE 5.4: Diagrams of reliability model of s_1 when it is expanded on common network components

It is noteworthy that the described approach for the calculation of job reliability is applicable only if all modules of the job reliability model, including predecessor jobs and incoming TT messages, are independent. However, it is quite likely that the reliability models of incoming messages and predecessor jobs share common network components. In such cases, the reliability of a job cannot be calculated directly from the reliability of series/parallel structures. Instead, first, the job reliability model needs to be expanded on every common network component. After that, the job reliability is computed based on different operational conditions of the common network components and the total probability formula as follows [126]:

$$Rel_{j_i} = Rel_{c_i} \cdot Prob\{job\ run\ correctly|c_i\} + (1 - Rel_{c_i}) \cdot Prob\{job\ run\ correctly|\bar{c_i}\} \quad (5.6)$$

where c_i presents a condition in which the component c_i is fault-free whereas $\bar{c_i}$ correspond to the faulty condition of component c_i.

Let's consider the reliability model of the first statement in Formula 5.5 (i.e. $Rel_{s_1} = Rel_{CCU_1} Rel_{m_1} Rel_{m_2}$) which is denoted in Figure 5.4a. As the diagram shows, there are four common network components (i.e. $ECN\ SW_4, l_4, ECN\ SW_2, l_5$) in this model. However, component $l_5, ECN\ SW_2$ are repeated in a series structure. Hence, only one of these components can be considered in the reliability model of s_1 as denoted in Figure 5.4b. After that, this reliability model is expanded on different combinations of the remaining common components (i.e. $ECN\ SW_4, l_4$) as shown in Formula [5.7-5.9].

$$Rel_{s_1} = Rel_{SW_4} \cdot Prob\{s_1\ works|SW_4\} + (1 - Rel_{SW_4}) \cdot Prob\{s_1\ works|S\bar{W}_4\} \quad (5.7)$$

$$Prob\{s_1|SW_4\} = Rel_{l_4} \cdot Prob\{s_1|SW_4,l_4\} + (1 - Rel_{l_4}) \cdot Prob\{s_1|SW_4,\bar{l_4}\} \quad (5.8)$$

$$Prob\{s_1|S\bar{W}_4\} = Rel_{l_4} \cdot Prob\{s_1|S\bar{W}_4,l_4\} + (1 - Rel_{l_4}) \cdot Prob\{s_1|S\bar{W}_4,\bar{l_4}\} \quad (5.9)$$

The diagrams [5.4c-5.4e] present the expanding of the reliability model s_1 on *ECN* SW_4 and l_4. The expansion process continues until the reliability model only comprises the independent modules, and thus the reliability can be calculated from the series/parallel structure. For example in s_1, the diagrams 5.4d and 5.4e which are derived from Formula 5.8 and 5.9, do not contain any common network components. Thereby, the reliability of these diagrams can be computed based on the reliability of the series/parallel structure.

This approach is applied to every statement that constitutes job reliability until the reliability model of the job does not encompass any dependent modules.

5.4.3 Reliability of Safety-critical System

In the previous sections, it is explained how to compute the reliability of message transmissions and safety-critical jobs which are considered as building blocks of the reliability model. The reliability of a safety-critical system is the probability of correct execution of all mission-critical jobs within the system. In other words, the reliability of a sink node in an application graph is defined as the system reliability. For instance, the reliability of the system in Figure 5.1 is:

$$Rel = Rel_{j_6} \cdot Rel_{m_{10}} \cdot Relj_7 \cdot Rel_{m_{11}}$$

As the reliability of all conditional edges that end at the sink node is one, the above equation can be written as follows:

$$Rel = Rel_{j_6} \cdot Rel_{j_7}$$

The reliability of jobs j_6 and j_7 can be computed recursively using the reliability of predecessor jobs, and incoming TT flows as it is shown in the previous section.

5.5 Fault-tolerant GA Scheduler

In this chapter, the Genetic Algorithm (GA) [22], which is explained in details in the previous chapter is extended to compute a transmission schedule of TT communication with the maximum system reliability. To optimize the reliability of the system, the reliability of safety-critical jobs that do not have any outgoing TT flows needs to be maximized:

$$\max(\prod_{\forall j \in J_1} Rel_j(t)) \quad (5.10)$$

In Formula 5.10, J_l comprises leaf nodes of a conditional application graph.

5.5.1 Genome Definition

In the fault-tolerant GA, each genome encompasses a set of genes. On the one hand, for job scheduling, one gene is assigned to every job. The job-specific gene determines on which end systems the job can run. On the other hand, for message scheduling, one gene is assigned to each TT message. The message-specific gene specifies the possibilities of the redundant route for that message. For encoding of a gene, a set of integers is used.

5.5.2 Population Initialization

The fault-tolerant GA generates initial individuals from the given system model, which leads to the first generation of individuals. The scheduler selects the best fitting individuals of every generation, mutates them based on a certain probability and then uses the simple-point crossover to create the next generation [143].

5.5.3 Fitness Function

To identify the best individuals, a fitness function is defined. This function calculates the reliability of the system for each individual as a fitness score. Then the scheduler chooses the individual with the best fitness score (i.e. the system reliability) for the next generation.

Algorithm 3 Fitness Function

1: **procedure** FITNESS(Genome g)
2: $E_{\text{TT.sorted}} \leftarrow$ *sort flows based on inter-dependencies*
3: $\forall e_i \in E_{TT.sorted}$:
4: S_{e_i}.processor $\leftarrow p \in S_{e_i}$*.CanRunOn job's genes*
5: Rec_{e_i}.processor $\leftarrow p \in Rec_{e_i}$*.CanRunOn job's genes*
6: $RedPath_{e_i} \leftarrow$ *opt redundant path using message genes*
7: $I_{e_i} \leftarrow$ find earliest feasible time slot
8: $A_{e_i} \leftarrow I_{e_i} + ED_{e_i}$
9: **if** $A_{e_i} > D_{Rec_{e_i}}$ **then return** 0
10: $Rel_{S_{e_i}} \leftarrow$ *calculate the reliability of sender job*
11: *SystemReliability* $\leftarrow$ *calculate the system reliability*
12: **return** SystemReliability

Algorithm 3 denotes the fitness function of the fault-tolerant GA. This function first sorts TT flows according to their precedence constraints. Then for every TT flow, the function allocates the available end systems to the sender and receiver jobs based on a job-specific gene in the genome (c.f. lines 4,5). The fitness function also determines the redundant path between the sender and the receiver end-system using the message-specific gene. To be more precise, the function first finds all possible redundant routes between sending and receiving end systems using the multiplication adjacency matrix procedure and then it selects the redundant path as a message forwarding route based on the message-specific gene.

After specifying the message forwarding routes, the fitness function finds the earliest feasible injection time of the TT messages considering the contention-free and the message-specific periodicity constraints. Then it calculates the arrival time of the message (A_{e_i}) to the receiver end system using the message's end-to-end latency (ED_{e_i}) which is defined in Chapter 4. If the message's arrival time exceeds the

deadline of the receiver job that means the violation of the delivery deadline constraint, the individual leads to an invalid solution. Therefore, the function returns 0 as a fitness score to prevent carrying over an unfeasible genome to the next generation. In the counter case, the function calculates the reliability of the sender job ($Rel_{s_{e_i}}$) as explained in section 5.4.2. After scheduling all TT messages, the function calculates the system reliability and returns it as a fitness score.

The fault-tolerant GA, apart from the advantages of the basic GA such as improvement of schedulability and resource utilization, aims to compute the TT transmission schedule with optimized system reliability. This optimization algorithm enhances the tolerance of time-triggered systems against crash failures at two levels: 1) This technique selects the forwarding routes whose constituent (including end-systems, switches, and links) are less probable to fail over time. 2) The scheduler also assigns a safety-critical job to the processing nodes which are less likely to encounter crash failures.

Algorithm 4 Fault-tolerant List Scheduler

```
1: procedure FAULTTOLERANTLISTSCHEDULER
2: Assign priority to each job
3: J_sorted ← sort jobs descending based on priorities
4: ∀j ∈ J_sorted is not scheduled:
5:     Scheduler(j)
6: SystemReliability ← calculate the system reliability
7: return SystemReliability
8: procedure SCHEDULER(Job j)
9:     if unscheduled job j has incoming flow then
10:        ∀e_i ∈ E_TT.incoming: Scheduler(S_e_i)
11:        is_pre_job_schedule ← true
12:    else if is_pre_job_schedule or job j has no child then
13:        for p ∈ j.CanRunOn do
14:            for e_i ∈ E_TT.incoming do
15:                RedPath_e_i ← find most reliable redundant paths
16:                I_e_i ← find earliest injection time
17:                A_e_i ← I_e_i + ED_e_i
18:                if A_e_i > D_j then go to next end-system
19:                Rel_j ← calculate job reliability
20:                Rel_e_i ← calculate message reliability
21: return
```

5.6 Fault-tolerant List Scheduler

The heuristic list scheduler presented in Chapter 4 is modified in a way that it meets the stringent timing requirements of the mission-critical applications while improving the overall reliability of the system. The list scheduler operates in two steps [144]: first, the priority of each job is computed, and after that, the jobs are mapped to the available end systems considering their priorities and precedence constraints. Likewise, the critical path is used in the heuristic list scheduler for calculating the priorities of the jobs. After the priority assignments, the scheduler sorts the jobs in descending order of their priorities and then schedules the jobs one by one from the highest priority to the lowest priority by assigning them to eligible end systems

($J.CanRunOn$). After mapping the job to the desired end system, the scheduler finds all possible routes between the sender and receiver end systems using the multiplication adjacency matrix and then permutes them to compute all the feasible redundant paths between the sender and the receiver. The reliability of the message forwarding is calculated for each redundant routes and the redundant routes with the best reliability value is chosen for TT message delivery. As the following step, the scheduler finds the earliest injection time of each incoming TT flow of the job considering the contention-free and the message-specific periodicity constraints. Then it computes the message's arrival time (A_{e_i}) using the end-to-end transmission delay (ED_{e_i}). Section 4.4 provides a detailed description of the end-to-end delay calculation. If the chosen time slot results in a message's arrival time (A_{e_i}) which exceeds the job's deadline (i.e. violating the delivery deadline constraint), then another end system is selected and the same procedure is repeated until all ingress TT flows are scheduled. Algorithm 4 gives a pseudo-code representation of the fault-tolerant list scheduler.

5.7 Experiments and Evaluation

The fault-tolerant GA and the heuristic list scheduler are developed in C++, and the experiments are carried out on a T460 ThinkPad computer with 32GB of memory and an Intel i5 CPU. The behaviour of the fault-tolerant GA and list scheduler are studied thoroughly using 625 different system models which are synthesized with the SNAP library [125].

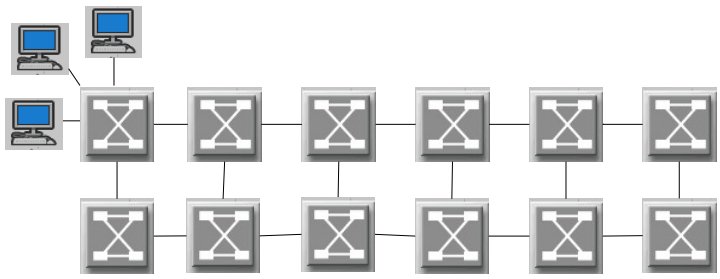


FIGURE 5.5: Grid network structure used in use case 1. Every switch is connected to 3 end-systems

For these experiments, five different use cases are used. The first three use cases aim to study the impact of TSN redundancy management on the overall system reliability, and each of these use cases contains 75 different system models. The fourth use case investigates the sensitivity of the system reliability to various network components through 200 synthetic system models. Similarly, the last use case utilizes 200 system models to study the impact of a varying degree of job redundancy on system reliability. The properties of the experimental use cases are listed in Table 5.1. The network structure in all use cases is the grid topology with a different number of devices, as described in Table 5.1. For example, Figure 5.5 presents the network structure of use case 1. Moreover, the conditional application graphs are generated in the form of a random Forest Fire directed graph [125]. In the experiment setup, the periodicity of TT flows is chosen from three different values (i.e. 50, 100 and 150 ms). Additionally, it is assumed that all switches and links in the experimental networks are identical. Therefore, the transmission time of each TT frame from one device to the neighbour device (T_{e_i}) is considered 40 µs since the processing delay

Use Case	1	2	3	4	5
Number of switches	12	14	16	12	6
Number of end-systems	36	56	80	36	18
Number of links	52	75	102	52	25
Number of jobs	12				6
Link/Device reliability distribution	0.991:10% 0.992:10% .. 0.998:10% 0.999:10%				0.993:25% 0.995:25% 0.997:25% 0.999:25%

TABLE 5.1: The properties of the experimental use cases

of all switches and also the transmission rate and the propagation delay of the links are similar. It is also assumed that all jobs have the same WET and deadline (i.e. 100 µs and 3.5 ms respectively). Besides the potential end systems for execution of a safety-critical job ($j.CanRunOn$) are selected randomly.

For the fault-tolerant GA, the number of generations is set to 100, the population size to 25, the crossover probability to 0.9 and the mutation probability to 0.2.

The first section of experiments illustrates the difference in the system reliability of the TT transmission schedules that are computed by the Fault-Tolerant HLS (FTHLS) and the Fault-Tolerant GA (FTGA) compared to the schedules generated by HLS and GA which do not support FRER. The test cases which are carried out for this purpose, are divided into three use cases (i.e. use case 1, 2 and 3) where each use case comprises different numbers of devices and links. The main goal of these use cases is to study the impact of different numbers of devices and links on the overall system reliability. In addition, each of the above use cases runs for three different numbers of TT messages (i.e. 15, 35 and 55 TT messages) to investigate the impact of network load on the overall system reliability. It is also assumed that all TT messages are essential in these use cases for the execution of successor jobs. For simplicity purposes and without loss of generality, it is assumed that the reliability of every network component remains constant for the duration of the application execution. As stated in Table 5.1, in this set of test cases, the reliability of devices and links alternates between 0.991 to 0.999 by the probability of 10%. For each use case, 25 synthetic system models are generated where each system model has different inter-flow dependency patterns and the grid network structure with the same number of devices and links as presented in Table 5.1.

As the diagram in Figure 5.6.a demonstrates, in use case 1 that runs for a different number of TT messages, the average of the system reliability of the transmission schedules which are computed by the FTGA is improved by 9% compared to the average system reliability of the schedules that are generated by the basic GA. Similarly, the FTHLS increases the average of the system reliability of the generated schedules by 9% compared to the HLS, which does not support the fault-tolerance features. The FTHLS and FTGA achieve a significant improvement in the system reliability by employing message duplication and transmissions of messages over the redundant paths. However, the HLS and the GA for every TT message only schedule one instance of the message over a particular route. This implies that the FTHLS and FTGA accomplish the reliability enhancement at the expense of makespan raise. The

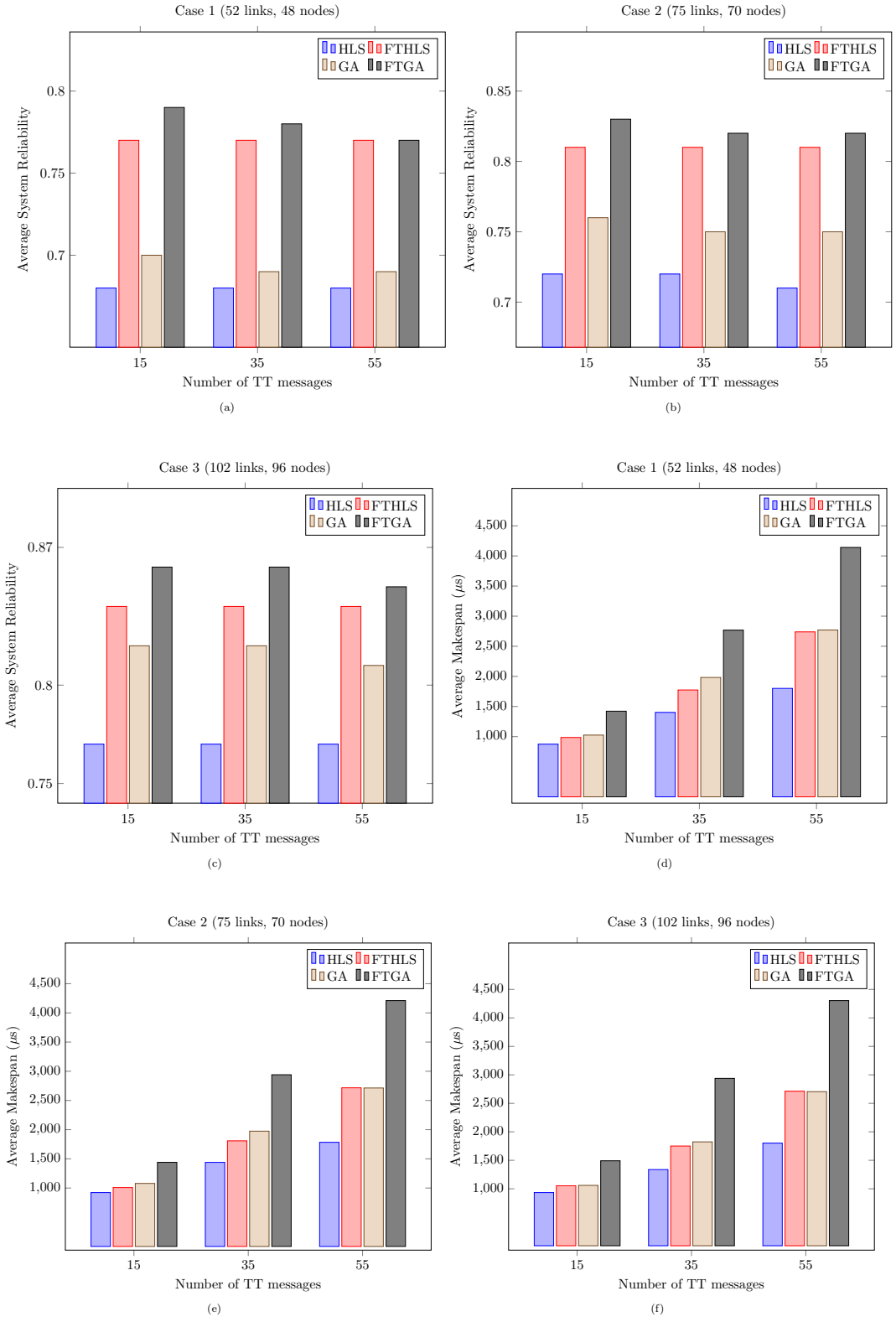


FIGURE 5.6: Average system reliability and makespan of schedules generated by FTHLS, HLS, FTGA and GA for different number of nodes, links and TT messages

average makespan of transmission schedules for the use case 1 is denoted in Figure 5.6.d. As the graph illustrates, the average makespan of the TT schedules, which are computed by the FTGA is increased by 42% compared to the average makespan of the schedules that are generated by the basic GA. Similarly, the FTHLS increases the average makespan of the generated transmission schedules by 30% compared to the basic HLS. Thereby the results show that the FTGA improves the system reliability of the transmission schedules by 9% at the expense of 42% makespan growth. The graph depicts a similar trend for the fault-tolerant HLS and the basic HLS where the FTHLS enhances the system reliability of the schedules on average by 9% at the expense of 30% makespan raise. Moreover, according to the graphs in Figure 5.6.a, when the number of TT messages increases while the number of devices and links in the network remains intact, the average of the system reliability decreases slightly.

The Figures 5.6.b and 5.6.e present the experimental results from the use case 2. As the graph in Figure 5.6.b depicts, the average of the system reliability of the schedules that are generated either by the FTGA or the FTHLS for different numbers of TT messages is improved compared to use case 1 mainly because the increased number of network elements in the grid network structure results in redundant routes with a higher reliability. Additionally, the diagrams (i.e. Figure 5.6.b) of use case 2 depict that the FTGA improves the average of the system reliability by 7% compared to the GA. To be more specific, in use case 2, the FTGA enhances the system reliability by 7% at the expense of 45% makespan raise. The graph, also, depicts that for the FTHLS the system reliability of the generated transmission schedules is improved by 9% at the expense of 28% makespan growth.

The simulation results of the use case 3 are demonstrated in Figures 5.6.c and 5.6.f. As shown in the diagrams (i.e. Figure 5.6.c), the average of the system reliability of the schedules that are computed by both the FTGA and the FTHSL for various numbers of TT messages is increased compared to use case 1 and 2. In use case 3, the FTGA enhances the average system reliability by 4% compared to the basic GA. Like in the former use cases, the FTGA achieves a system reliability improvement at the expense of 53% makespan growth. Similarly, the FTHLS increases the average system reliability of the transmission schedules by 7% at the expense of 30% makespan raise.

It has to be noted that in all above use cases which are carried out for different loads of networks the FTGA outperforms the FTHLS in terms of the system reliability. Since the FTGA during the scheduling process always selects the redundant paths for forwarding TT messages, it leads to an optimized overall system reliability. However, the FTHLS for each TT flow chooses the most reliable redundant routes between sender and receiver end systems which results in neglecting the inter-flow dependencies and the collective impact of the message reliabilities in the system reliability.

Scheduler	use case 1 (s)	use case 2 (s)	use case 3 (s)
HLS	8.46	26.13	56.87
FTHLS	9.02	29.23	60.9
GA	172	134	125
FTGA	450	719	5317

TABLE 5.2: Average execution time of HLS, FTHLS, GA and FTGA for use cases 1-3.

Furthermore, as presented in Table 5.2, the average execution time of the FTGA is significantly longer than the basic GA since the FTGA needs to explore a bigger search space of the legitimate fault-tolerant schedule possibilities. On the other hand, the FTHLS solves the scheduling problem in a slightly longer time than the HSL mainly because it explores a bigger system implementation design space. It is noteworthy that the average solving time of the FTGA is considerably longer than the FTHLS in similar use cases. Nevertheless, in the mentioned test cases, the average of the system reliability of the schedules generated by the FTGA is higher than the system reliability of the schedules computed by the FTHLS.

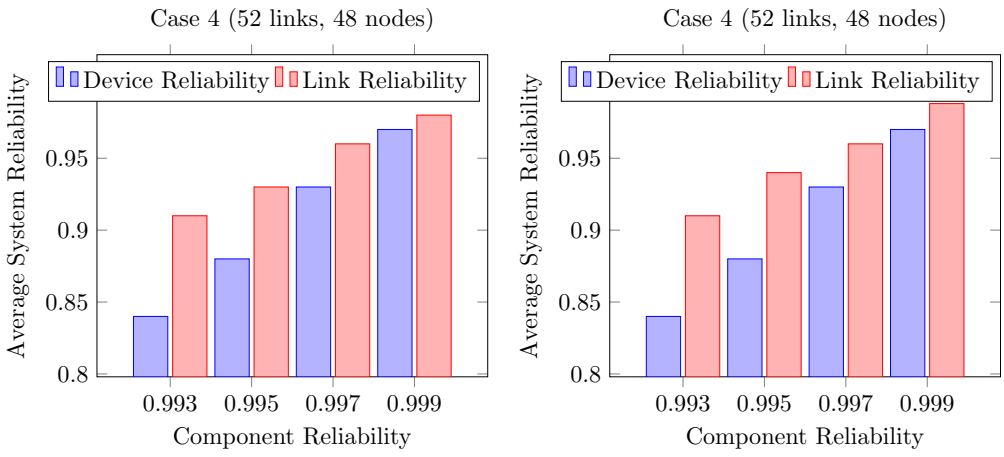
Scheduler	Use case 4 Avg. Makespan (μs)	Use case 4 Avg.Exec time (s)	Use case 5 Avg.Makespan(μs) (μs)	Use case 5 Avg.Exec time (s)
FTLS	1024	8.5	933	0.99
FTGA	1420	50	1254	791

TABLE 5.3: Average execution time and makespan of the schedules generated by FTHLS and FTGA for use case 4-5.

Our reliability model identifies which network components have more impact on the overall system reliability. The second part of the experimental results aims to evaluate the sensitivity of the system reliability to different network components. To achieve this goal, two sets of experiments are conducted based on the use case 4 set up that is described in Table 5.1. In the first set of experiments, it is assumed that all physical links are fault-free while the network devices, including end systems and switches, are failing over time. In the counter set of test cases, it is assumed that the devices are fault-free, although the communication links encounter crash failures over time. Namely, in the first set of experiments, the reliability of links is set to 1 ($Rel_l = 1$) while the different values of reliability for network devices are considered (i.e. 0.993, 0.995, 0.997 and 0.999). In the opposite set of test cases, the device reliability remains constant ($Rel_{sw}, Rel_{\text{es}} = 1$) while the link reliability varies from 0.993 to 0.999. For each set, as mentioned earlier of test cases, 25 synthetic system models like in the first section of experiments are generated.

As the graph in Figure 5.7.a denotes, for FTGA when the reliability of a link is increased, the average reliability of the system is improved by 4% compared to the test cases where the same growth happens for the device reliability. The diagram 5.7.b denotes a similar improvement (i.e. 4%) in the system reliability of the transmission schedules that are computed by FTHLS. Therefore, according to the experimental results, the reliability of a link plays a key role in the average reliability of our synthetic system models. This implies that the more reliable physical links lead to transmission schedules with higher system reliability. Nevertheless, both FTGA and FTHLS offer the same average of the system reliability for these sets of experiments, but according to Table 5.3 the FTGA generates the transmission schedules with a longer makespan compared to the FTHLS. In use case 4 like the previous use cases, the FTGA takes a longer time to find the schedules with the optimal system reliability compared to the FTHLS. The average execution time and makespan of the schedules which are generated for use case 4 and 5 by FTHLS and FTGA are presented in Table 5.3.

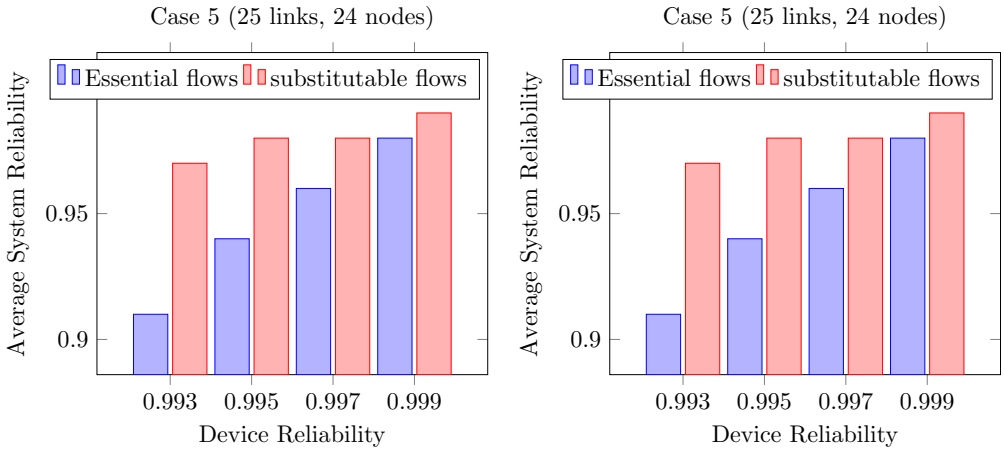
The third part of the experimental results has studied the impact of a varying conditions of the control transfer in forms of TT messages between mission-critical jobs. For this purpose, two sets of test cases are considered based on the use case



(a) FTGA (b) FTHLS

FIGURE 5.7: Average of system reliability of schedules generated by FTGA and FTHLS for different component reliability

5 set up. In both sets of test cases, links are fault-free while the device reliability alternates between values of 0.993, 0.995,0.997 and 0.999 by the probability of 25%. In the first set of test benches, every process with incoming edges requires the TT messages from all its predecessor jobs for execution. However, in the second set of experiments, the condition of all control transfers between mission-critical jobs is substitutable. Like the previous experimental sections, for every use case, 25 system models are generated.



(a) FTGA (b) FTHLS

FIGURE 5.8: Average of system reliability of schedules generated by FTGA and FTHLS for different conditional precedence constraints

According to the graphs in Figure 5.8.a, the average of the system reliability of the transmission schedules generated by FTGA for the latter set of experiments where all TT flows are substitutable, is enhanced by 4% compared to the first set of test cases in which all TT flows are essential. The diagram 5.8.b illustrates the same enhancement in the system reliability of the schedules computed by FTHLS

for the mentioned test cases. However, the FTHLS improves the average makespan of the schedules compared to the FTGA. Hence, the experimental results of both schedulers (i.e. FTGA and FTHLS) show that the higher level of redundancy for the safety-critical jobs results in more reliable TT communication. According to Table 5.3 the FTHLS outperforms the FTGA in terms of the average makespan and the execution time, although both schedulers accomplish the same degree of reliability in the transmission schedules computed for use case 5.

Chapter 6

Time-Sensitive Networking Simulation Framework

As described in chapter 1, the virtualized integrated system hosts several critical and non-critical modules and offers these functions the necessary resources and services through well-defined software abstraction layers [149]. Moreover, the communication layer of the virtualized integrated system needs to provide reliable network services, mixed-criticality traffic and the simplified integration and configuration management. For this reason, the communication layer of the virtualized integrated system is defined based on the principles of the TSN standard, which is the most recent real-time Ethernet extension and supports the requirements mentioned earlier. Networking experts and technology manufacturers use simulation frameworks extensively to emulate and validate new networking solutions mainly because the implementation and deployment of novel protocols on hardware is a very time consuming and costly process. Since TSN is chosen as a networking solution for the virtualized integrated system, this work presents a simulation framework for TSN, which is developed as an Ethernet-based network for mixed-critically traffic. To this end, firstly the simulation models implement the time-based features of TSN, including the time-aware shaper and a policer in the Riverbed Modeler framework. A model of the TSN-aware device uses the standard MAC unit for switching messages, however, at the same time it adds the necessary functionalities to support strict temporal requirements. The described implementation is modular and can be easily integrated into different vendor-specific network elements. The evaluation of the time-based features of TSN is performed using several use cases and network configurations, and the simulation results are compared against the temporal constraints of safety-critical applications.

Further, the TSN models, as described earlier, are extended to support different FRER functionalities since the fault tolerance capability of TSN is essential for message exchanges between mission-critical applications. The FRER functions are implemented in a modular manner so that they can seamlessly be integrated into the existing TSN models at appropriate stages. Moreover, this work presents a fault injection model to verify the correctness and applicability of the FRER module. The fault injector model simulates different faulty behaviours (including transient and permanent errors) in the emulated TSN network. The experimental results are used to evaluate the impact of the FRER module on the reliability of TT communication.

Aside from TSN time-aware features (i.e. time-based filtering, policing and shaping) and a non-time-aware feature (i.e. FRER), the model of a TSN-capable device implements the TSN clock synchronization mechanism. Namely, the TSN-aware model uses TSN synchronization procedures instead of the static reference time to enforce time-based features of TSN. As a result, this simulation model provides an opportunity to evaluate the correctness of the real-time capability of TSN solutions

in more realistic networking scenarios. Additionally, the time-aware system model aims to study the correctness and applicability of the Best Master Clock Algorithm (BMCA), the delay measurement and the synchronization process which are introduced in the IEEE 802.1AS-Rev protocol mainly because this sub-protocol is not finalized yet.

A time-aware system requires a local clock with high accuracy to guarantee the deterministic delivery of time-triggered traffic. Therefore, this work models the local clock of each time-aware system with different drift rates (including linear and non-linear drift rates). Furthermore, the behaviour of the time synchronization modules is studied in the presence of crash failures. For this purpose, first, the re-execution of BMCA is emulated by failing either the primary grandmaster node or the directly attached link using the fault injector model. After that, the accuracy of the device's clock is evaluated during both the grandmaster failure and the handover period.

A TSN-aware model needs to be configured to employ TSN features. For this reason, this work develops a fully centralized TSN model that aims to provide a method for dynamic configuration of TSN-capable devices. In the fully centralized configuration model, a central network configurator firstly acquires knowledge about the network topology and the associated traffic profiles and then computes the schedules for the real-time applications and the corresponding TT traffic using the received information. In the end, the central entity remotely configures the TSN-capable devices so that their temporal and reliability requirements are met.

To sum up, this work develops the simulation models with both time-based and non-time-based services of TSN. Consequently, it provides a comprehensive simulation platform for modelling, performance and reliability evaluation of TSN networks.

The rest of the chapter is structured as follows: In section 6.1, related work on time-triggered simulation frameworks is discussed. Section 6.2 presents the conceptual models used in the TSN simulation framework. Additionally, this section details the implementation of the described models in the Riverbed simulation framework. In the last section, the correctness and applicability of TSN services, including configuration, redundancy management, clock synchronization, time-aware scheduling and filtering are studied through several test scenarios. The simulation scenarios are carried out based on an example train network layout to obtain realistic experimental results.

6.1 Related Work

For the evaluation of deterministic networks, a wide range of simulation frameworks has been developed. Alderisi et al. [150] use the OMNET++ simulation framework to show that an AVB in-vehicle network with a double-star structure fulfills timing requirements of automotive applications including infotainment and Advanced Driver Assistance Services (ADAS). Zinner et al. [151] discussed the problem of co-existence of the AVB networks with the legacy vehicular network technologies such as MOST and FlexRay mainly because in practice the transformation of an automotive network to the fully AVB-compliant network happens incrementally. For this reason, the authors introduced a method to map the QoS services offered by MOST and FlexRay to the predefined priorities in an AVB network. Further, they confirmed the viability of co-existence of legacy technologies and AVB-aware devices within an automotive network using a simulation framework.

Steinbach et al. [152] implemented a simulation framework called INET-Framework for TTEthernet in OMNeT++. The same authors in [153] examine the suitability of AVB and TTEthernet technologies for time-triggered communication within a vehicular network using the extended INET-Framework. The empirical results illustrated that both standards offer temporal isolation required by automotive traffic. However, the performance of TTEthernet is less affected by background traffic than AVB. Alderisi et al. [154] provided the same results regarding performance comparison of AVB and TTEthernet for in-vehicle networks. The authors in [86] extended the INET-Framework with the AVB standards to evaluate the effect of co-existence of the time-critical flows and the AVB stream classes on the shared physical infrastructure. This framework was extensively used to analyze the impact of AS6802 (TTEthernet) and AVB on the real-time applications in the automotive use cases. Furthermore, several works such as Kawahara et al. [155] and Tuohy et al. [156] focus on the performance evaluation of the AVB solution in vehicular networks using simulation models in OMNeT++ and ns-3 respectively. Imtiaz et al. [157] investigated the applicability of AVB scheduling mechanism for industrial systems. For this purpose, they conducted a test scenario where the transmission of a lengthy BE frame has an overlap with the dissemination of AVB traffic. The simulation results illustrated that in such use cases, the AVB standard is unable to offer deterministic transmission for audio and video traffic. The same authors in [158] addressed this issue by a new preemptive shaping approach whereby a conflicting large BE frame is preempted, and its transmission will be resumed after sending the AVB traffic. Lim et al. conducted extensive research on the suitability of the AVB standard for vehicular networks [159, 160]. For instance, in [159] the authors developed an AVB network simulator using OMNeT++ whereby they demonstrated that in the simulated network the delivery latency and jitter of AVB traffic classes significantly improved compared to the IEEE 802.1Q-based network. On the contrary, the simulation results showed that the transmission of control data as the highest 802.1Q priority frame offers better performance in terms of end-to-end delay, jitter and packet loss compared to the AVB networks.

Results derived from the simulation frameworks mentioned above validate the conclusions drawn from mathematical analysis, stating that AVB does not meet strict timing requirements necessary for mission-critical traffic [157, 161, 159].

This chapter develops the simulation model of 802.1Qbv and Qci capable devices (including end systems and switches) to evaluate the differences between the scheduling method of TSN and the credit-based shaping of AVB. Several studies recently focus on different aspects of the new scheduling mechanism proposed in TSN. The authors in [96] discussed the key parameters that impact the solution for the TSN scheduling problem. Craciunas et al. [113] listed the scheduling constraints for IEEE 802.1Qbv and provide them to Satisfiability Modulo Theories (SMT) and Optimisation Modulo Theories (OMT) solvers to synthesize the valid Gate Control List (GCL), which provides a bounded end-to-end delay and delivery variation of safety-critical traffic. In [79], Pop et al. proposed the optimized configuration approach for the scheduled networks. This work used the ILP approach to synthesize the GCL for each egress port so that all TT streams are schedulable and the minimum number of queues are dedicated to the TT flows to satisfy their timing requirements. In [162], Park et al. using the OMNET++-based simulation framework ensured that TAS provides deterministic delivery of TT messages in the presence of AVB streams and BE traffic. However, they only studied a limited aspect of the TAS mechanism.

Similarly, the authors in [163] implemented a simulation model for an 802.1Qbv capable switch that shapes outgoing traffic based on predefined GCLs. This work validated that the IEEE 802.1Qbv standard offers deterministic behaviours for mission-critical systems. Nasrallah et al. [164] carried out extensive studies on the Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS) that are introduced by the TSN task group. For this purpose, the authors proposed two new methods (i.e. Adaptive Bandwidth Sharing (ABS) and an Adaptive Slot Window (ASW)) to enhance the performance of standard TAS. Both of these mechanisms ensured that TT traffic meets their stringent timing constraints while improving the quality of service of the lower priority traffic (e.g. BE traffic). Additionally, this work showed that ATS outperforms TAS for sporadic streams. Nevertheless, TAS, unlike ATS, provides adequately low latency delivery for TT traffic regardless of the number of conflicting lower priority messages.

The IEEE 802.1CB protocol is also modelled in this chapter to address the reliability requirements of cyber-physical systems. Over the last years, many works studied the suitability of different fault-tolerant techniques for time-sensitive systems. Kehrer et al. [165] delivered a detailed comparison between two different fault-tolerant methods of TSN networks (i.e. a decoupling stream reservation and redundancy protocol and also a harmonized stream reservation and redundancy protocol) where both approaches aim to offer the deterministic reconfiguration time interval. In [166], Qian et al. firstly provided a comprehensive overview of the IEEE 802.1CB standard and then emulated an 802.1CB-aware network in visual studio. The authors in [167] introduced a Priced Timed Automata (PTA) model to determine potential failures in the TSN networks. Thereby, they used the PTA to assess the fault-resilience of the transmission schedules, particularly under Single Event Upsets (SEUs). Prinz et al. [168] integrated the implementation of the TSN redundancy method to the Industry 4.0 (I4.0) framework that was developed previously by the same authors [169]. The empirical results verified the fault-resilient of the TSN redundancy mechanism against different network failures in the I4.0 framework.

IEEE 802.1CB is a spatial-based redundancy protocol. Therefore, it is not cost-efficient to mask transient and intermittent faults which can frequently happen due to environmental changes with TSN spatial redundancy mechanisms. To address this issue, Alvarez et al. [170] designed a time-based redundancy mechanism called Proactive Transmission of Replicated Frames (PTRF) in which multiple copies of each TT message are sent over the same forwarding path within a single transmission period. As a result, even in the presence of transient failures, at least one copy of the message is delivered to the destination. The authors additionally combined the PTRF mechanism with the TSN spatial redundancy method to enhance the overall system reliability while optimizing network utilization.

Over the last decades, the focus of several works were clock synchronization procedures of real-time systems. The authors in [171] provided a detailed description of the synchronization process for both logical and physical clocks. Lundelius et al. [172] investigated the factors that practically have an impact on the precision of the synchronized global time. Besides, the fault-tolerant clock synchronization mechanisms are utterly discussed in [173, 174]. Stanton et al. [175] provided a comprehensive overview of the TSN clock synchronization procedure which plays a critical role in cyber-physical systems.

In recent years, simulation tools are extensively used to assess the performance of IEEE 1588 and IEEE 802.1AS. Hao Guo [176] emulated a PTP-aware network using the OPNET simulation framework. However, the simulated PTP-aware system was vastly simplified, comprising only two boundary clocks and one end-to-end

transparent clock. Additionally, this work assigned the role of each device statically. As a result, the simulation framework was only used to evaluate the delay measurement and the synchronization processes. Furthermore, the authors evaluated the impact of using a regular switch instead of an end-to-end transparent clock on the synchronized accuracy. The simulation results showed that the synchronization error is substantially raised when the load of cross traffic increases mainly because in such cases a standard Ethernet switch measures the residence time of PTP messages inaccurately. In [177], the author developed simulation models for IEEE 1588 in the OMNeT++ framework to study the interdependence of different configuration parameters and the synchronization precision. According to the experimental results, the selected periodicity of synchronization messages has a significant impact on the precision of the synchronized time. In more detail, the simulated network containing 50 nodes achieved a synchronization precision of 0.7 µs when the synchronization interval was set to 31.25 ms

Garner et al. conducted numerous studies on the IEEE 802.1AS standard [178, 179, 180, 181]. Namely, they assessed the performance of different functionalities of IEEE 802.1AS through both simulation and a real-life prototype. However, these evaluations are only limited to small-sized networks and also ignored the synchronization error such as physical layer communication jitter. Lim et al. [160] presented an OMNeT++-based simulator which aimed to emulate automotive networks supporting the IEEE 802.1AS standard. The authors simplified the implementation of AVB clock synchronization by using the static assignment of clock roles, excluding external clock sources and neglecting measurement of the phase and frequency discontinuity. Consequently, this work evaluated only the performance of IEEE 802.1AS peer delay measurement and synchronization process. The empirical results showed that the measured peer delay using two different sampling filters converged to the actual propagation delay (i.e. 40 ns) within an error range of $\pm$10 ns. For evaluation of the synchronization process, a daisy-chain based topology comprising time-aware and non-time-aware systems was simulated. Then the precision of clock synchronization was measured for two different synchronization intervals (i.e. 62.5 ms and 125 ms). For a synchronization interval of 125 ms, the simulated network with 100% load of background traffic achieved the synchronization precision of 1 µs. On the other hand, a synchronization interval of 62.5 ms decreased the synchronization error by 50% compared to a synchronization interval of 125 ms. Gutierrez et al. [182] unlike the works as mentioned above investigated the impact of different sources of synchronization errors like the clock granularity and the jitter of the communication layer especially for larger time-sensitive systems since the synchronization error is accumulated while transporting timing information across the large network.

Le et al. [183] modelled the time-aware shaping mechanism in OMNeT++ simulation framework. They additionally emulated the TSN clock synchronization method to enable more accurate evaluation of TSN time-aware features. However, this work did not provide any information on the implementation details and operation of different functionalities of IEEE 802.1ASRev. In [184], the authors discussed how to integrate the TSN services to the legacy industrial networks. For this reason, they presented the simulation framework that modelled the Sercos as an example of legacy technologies and TSN features in a single network.

This chapter details TSN synchronization modules which are implemented on top of our TSN simulation models that support IEEE 802.1Qbv, IEEE 802.1Qci and IEEE 802.1CB. Unlike the described studies, this work is not limited to the modelling of TSN delay measurement and synchronization mechanisms but it also dynamically assigns the role of each device's port by executing BMCA. In contrast to

existing clock synchronization simulation frameworks, this simulator considers network faults in real-time systems. To this end, various faulty behaviours such as crash and link failure are simulated within the synchronized network. Then the impact of each failure on the performance of the clock synchronization is assessed thoroughly.

Several recent studies [185, 186, 187] discussed how the Software Defined Network (SDN) concept could facilitate the complex configuration process of real-time systems. The authors in [188] developed the first SDN-based prototype for a deterministic Ethernet network. This proof-of-concept which used Powerlink Ethernet and the OpenFlow protocol showed how SDN reacts to link failures in real-time systems.

[189, 190] presented complete configuration solutions for industrial networks based on the fully centralized model introduced in IEEE 802.1Qcc. These solutions examined the feasibility of the integration of the machine to machine communication protocol (OPC UA) to TSN networks in a way that the timing and reliability requirements of mission-critical systems are met. In [190], the authors proposed the Configuration Agent, which enables self-configuration of TSN features particularly transmission schedule tables within real-time systems. This work additionally studied protocols such as YANG, NETCONF and OPC-UA which are used for encoding and transporting the configuration information. The authors further proposed the enhancement to these protocols so that they can fulfill the requirements of real-time networks. However, in the mentioned works, the authors did not validate their proposal through any proof-of-concept. The same authors in [191] proposed the TSN standard as an appropriate communication solution for cyber-physical systems which use fog computing. The framework in [190], benefited from a configuration agent to support dynamic configuration of network components, particularly configuration challenges related to scheduling. Said et al. [192] introduced a more comprehensive SDN-based framework for configuring different TSN services, particularly clock synchronization. Besides, they verified the proposed solution through a proof-of-concept.

This chapter, aside from modelling a device with real-time and reliability capabilities, describes the implementation of a fully centralized configuration model and other necessary functionalities which are specified in IEEE 802.1Qcc for dynamic and remote configuration of TSN-capable devices. Since the IEEE 802.1Qcc protocol restricts the system designers to neither a specific network management standard nor a particular network discovery mechanism, this work uses the NETCONF protocol for managing the network and the Rapid Spanning Tree Protocol [49] for topology changes detection. This simulation framework provides an opportunity to analyze the performance of the configuration process, which is measured as the time interval required for the convergence of the application and the network after initiating configuration. Therefore, this work indicates whether the centralized configuration model which uses RSTP messages for setup forwarding paths and transmits the NETCONF messages as best-effort traffic is suitable for the use in an actual TSN system.

To sum up, this chapter introduces a simulation framework, which consists of simulation models for both time-based and non-time-based services of TSN. Thereby, this TSN simulator is different from the prior TSN simulation frameworks that focused only on specific TSN features. TsimNet [145] is one of the most recent TSN simulation frameworks which is implemented on top of the INET framework. This work addressed TSN sub-protocols which are not time-based such as frame preemption, frame replication and elimination and per-stream filtering mainly because TsimNet is aimed to evaluate TSN standards for the domain of avionics, where no

time synchronization mechanism is used till now due to the certification overheads. Consequently, the TSN simulator proposed in this chapter offers a more comprehensive simulation platform for modelling, performance and reliability evaluation of TSN networks.

To perform an accurate evaluation, this work conducts experiments using a real-life use case based on a train communication network.

6.2 TSN Simulation Framework

It is essential to evaluate and validate the applicability of TSN solutions for the communication layer of the virtualized integrated system. The manufacturing and verification process of switches and end-systems which are compliant to the TSN standard is expensive and time-consuming. As a result, simulation tools are considered as a cost and time-efficient options for analyzing the temporal and non-temporal attributes of TSN. The TSN features are evaluated through different network performance metrics such as end-to-end delay and jitter. The simulation models provide an opportunity for industrial manufacturers and operators to simulate various industrial network topologies and use cases at low cost and with high precision (e.g. in order of nanoseconds).

However, there are a large number of network simulation frameworks in the market. NS-2, OMNeT++ and OPNET simulators are among the most popular simulation tools. The Network Simulator-2 (NS-2) [193] and the Objective Modular Network Testbed in C++ (OMNeT++) [194] are both open-source discrete-event triggered simulation tools. On the other hand, Riverbed Modeler (formerly called OPNET) [24] is a robust commercial simulation framework which is widely used in industry and academia to simulate and evaluate different communication layers, network elements and protocols mainly because it provides a reliable and robust evaluation of the simulated system. Due to widespread usage, Riverbed Modeler evolves continuously to support a broader range of network protocols and technologies. Besides, Riverbed Modeler has a comprehensive Graphical User Interface (GUI) that enables the user to model a network topology on different layers (e.g. physical layer). The visual design of a network topology maps to the real system implementation using an object-oriented programming approach. Furthermore, Riverbed Modeler is a discrete-event triggered simulation tool, meaning that when a user develops a use case, the events are used to simulate the system operation. This simulator also offers a programming technique to implement user-defined network protocols and message formats.

The Riverbed's core functionalities are modelling, simulating, and analysis. In Riverbed Modeler, the simulation results are presented in different readable forms (e.g. graphs, statistics). Some of Riverbed's capabilities based on the OPNET white-paper can be listed as follows: parallel and event-triggered simulation kernel, powerful GUI for model development, user-friendly debugging and data analysis tools, discrete event simulation engine, several standard components with source code, object-oriented modelling and open interface for importing external models [195].

To develop a customized network model in Riverbed, a user needs to specify the simulation network, node and process models. The network model simulates a distributed communication system using standard and customized components. The node model represents network devices and consists of different modules. In a node model, modules communicate with each other via packet streams. Each module is defined using one or more process models. A process model is implemented

by Finite State Machines (FSM) and can be configured via process interface. An FSM specifies the behaviour of the module through a set of states and conditions [66]. Finite state machine models are programmed in C and C++ [196].

This work uses Riverbed simulator for modelling the communication infrastructure of the virtualized integrated system because Riverbed Modeler is known as a robust simulation framework for the modelling and performance evaluation of a wide range of communication systems. More specifically, the Riverbed Modeler provides a platform to build upon already developed models for different time-triggered protocols to satisfy the timing and reliability requirements of the virtualized integrated system.

```
<!-- Definition of the switch configuration -->
<SwitchConfiguration>
<node name="TSN_switch1" min_match_score="
   strict_matching" ignore_questions="true"
   model="ethernet16_switch_adv_tsn">
<ext-attr name="config_file" type="string">
<default-value value=""/>
</ext-attr>
<ext-attr name="CGL_file" type="string">
</ext-attr>
<attr name="BridgeParameters.count" value="1"/>
<attr name="BridgeParameters[0].QoSParameters.
   count" value="1"/>
<attr name="BridgeParameters[0].QoSParameters
   [0].QoSSupport" value="Enabled" symbolic="
   true"/>
<attr name="BridgeParameters[0].QoSParameters
   [0].DefaultPortQoSScheme" value="
   Strict_Priority" symbolic="true"/>
<attr name="Switch_Port_Configuration.count"
   value="16"/>
<attr name="config_file" value="config"/>
<attr name="CGL_file" value="CGL"/>
</node>
...
</SwitchConfiguration>
<!-- Definition of the link configuration -->
<LinkConfiguration>
<link name="TSN_switch_3-Server_2"
   min_match_score="strict_matching"
   ignore_questions="true" model="100
   Gbps_Ethernet" destNode="Server_2" srcNode="
   TSN_switch3" class="duplex">
<attr name="transmitter_a" value="TSN_switch3.
   hub_tx_1"/>
<attr name="receiver_a" value="TSN_switch3.
   hub_rx_1"/>
<attr name="transmitter_b" value="Server2.
   hub_tx_0_0"/>
<attr name="receiver_b" value="Server_2.
   hub_rx_0_0"/>
<attr name="doc_file" value="nt_link"/>
<attr name="tooltip" value="
   Ethernet_100Gbps_Link"/>
</link>
...
</LinkConfiguration>
```

FIGURE 6.1: An example of network setup XML file

This chapter presents a TSN network simulator and the associated models of switches and end-systems which implement the subsets of mechanisms proposed in the TSN protocol-suites. Therefore, these TSN simulation models can be used to simulate several industrial use cases at a network level. This network simulator particularly supports the time-based (IEEE 802.1Qbv and Qci) and non-time-based (FRER) features of TSN. Furthermore, all components in the TSN simulator implement IEEE 802.1ASRev clock synchronization mainly because TSN-aware devices use the global time to police and schedule mixed-criticality traffic. The TSN simulator also contains a central entity which is responsible for the configuration of TSN-capable devices.

```
"platform": {
   "nodes": [
      {
       "id": 0,
       "is_ETBN": false,
       "is_CS": false
      },
      {
       "id": 1,
       "is_ETBN": false,
       "is_CS": false
      },
      {
       "id": 2,
       "is_ETBN": true,
       "is_CS": false
      }
   ],
   "links": [
      {
       "start": 0,
       "end": 2
      },
      {
       "start": 1,
       "end": 2
      }
   ]
```

FIGURE 6.2: An example of network setup JSON file

6.2.1 Network Generator Model

In Riverbed, there is a possibility to create an arbitrary network topology by importing an XML file. This XML file describes all node models which form a simulation network and the connections between them. This feature allows defining various industrial use cases without accessing Riverbed's GUI, selecting network devices one by one and making a connection between them. Consequently, several industrial

networks can be imported to Riverbed and simulated in a time-efficient manner. The example of a network XML file is presented in Figure 6.1.

To simplify the process of defining a simulation network, a network topology generator model is defined which gets the description of network components and physical links between them in JSON format and converts them to the XML format. As shown in Figure 6.2, the JSON file only contains the high-level definition of the network topology. Therefore, the network topology generator sets several parameters of node models to the default values. This approach, however, makes the creation of a simulation network faster and independent from the node model specifications, but it limits the flexibility of network setup definitions.

6.2.2 Configuration of TSN Devices

The TSN switch model integrates time-based ingress policing and time-aware egress shaping into the existing queuing and scheduling methods of the standard switch. Besides, in the TSN simulation framework, the conventional end systems are modified to dispatch and receive different types of messages based on predefined traffic profiles. Therefore, each TSN-capable node model in the TSN simulator requires the profiles of ingress traffic and the port-specific GCLs to police incoming traffic and shape outgoing frames at certain time instants. In TSN networks, there are three types of traffic: time-triggered, rate constraint and best effort. Each TT traffic profile comprises the following parameters:

- **Source MAC address**: It specifies the MAC address of the end system that sends frames.
- **Phase**: It defines a time instant at which a TSN device expects that the reception of the TT flow starts. This value is an offset in the range of [0, flow's period].
- **Period**: Each TT stream is received and transmitted periodically. This value defines the period time.
- **Transmission window**: In TSN, a TT flow can consist of more than one Ethernet frame. Thus, this parameter specifies how long the arrival of TT frames can continue.
- **VLAN ID**: It determines a VLAN identifier for the IEEE 802.1Q header.
- **Destination ports**: For a TT flow, apart from the arrival time, the route from a sender to the receiver is specified offline. Therefore, this parameter lists egress ports.

Aside from the TT traffic profile, there are non-TT traffic profiles which contain Rate Constrained (RC) traffic parameters. The RC traffic is identified with attributes similar to TT stream parameters. The only difference is related to the period field. For RC traffic, this field is called BAG and specifies the minimum time interval between two consecutive frames which belong to the same RC traffic.

In a TSN-capable device, when the incoming frame attributes do not match any TT or RC traffic profile, the packets are classified as BE traffic because they do not have any timing constraints. This is the reason why the BE traffic parameters are not defined as a part of non-TT traffic profiles. However, the BE traffic attributes are provided to the end-system model, which is responsible for generating BE messages.

It has to be noted that the GCL is specified for each egress port separately and contains the following parameters:

- **Queue mask**: This attribute specifies the state of each queue's gate in the time interval between **start time** and **end time**.

For instance, consider 10000000 as a queue mask while start and end time are set to 0 and 10 microseconds respectively. This parameter means the gate of queue number 7 would be open and enabled to transmit traffic at any time between 0 and 10 microseconds. All other queues are closed in the defined period. Each port-specific GCL runs over a period that is assigned to the Least Common Multiple (LCM) of periods of all the TT flows destined to that port. Furthermore, all TSN devices start running GCLs at the same time.

The port-specific transmission schedule of a TSN device (i.e. GCL) can be generated using any of schedulers that are introduced in Chapter 4 and 5. However, in the TSN simulator the Fault-Tolerant Heuristic List Scheduler (FTHLS) is utilized to calculate port-specific GCLs. FTHLS takes a description of the network structure and a specification of the jobs that execute on the network and associated TT messages as inputs. In the output, FTHLS specifies the end system that each job should run on and also the routes that each TT message should traverse to the receiving end system. FTHLS additionally calculates the injection time at which each job starts the transmission of corresponding TT messages.

In the TSN simulation framework, the configuration model, which is developed according to the TSN fully centralized configuration model manages components residing in the network through the NETCONF protocol. Moreover, a central configuration entity acts as both CNC and CUC, meaning that it collects data about the capabilities and requirements of both switches and end systems and further configures these components accordingly. The reason behind this design decision is to avoid additional communication overhead between these two modules.

6.2.2.1 Modeling TSN Configuration Parameters

It is necessary to model the TSN configuration parameters introduced above with the YANG modelling language. These models are used to encode the configuration information related to switches and end systems which can be modified using NETCONF RPCs. Figure 6.3 and Figure 6.4 present the YANG models which are designed for formatting the configuration parameters of TSN switches.

```
        grouping groupStreamConfiguration {

          leaf num_TT {type uint32;}

          list TT {
            key "src_address,vlan_id";

            leaf src_address {type int64;}
            leaf phase {type decimal64;}
            leaf period_duration {type decimal64;}
            leaf transmission_duration {type decimal64
               ;}
            leaf vlan_id {type vlanid;}
            container dest_ports {
              leaf num_dest_ports {type uint16;}
              leaf-list port_id {type uint16;}
            }
          }
        }
```

FIGURE 6.3: YANG model of the configuration data responsible for stream scheduling in a TSN switch

```
grouping groupGCLPortConfiguration {
  list GCL {
    leaf num_GCR {type uint32;}
    list GCR {
      leaf start_time {type uint64;}
      leaf end_time {type uint64;}
      leaf queue_mask {type queuemask;}
    }
  }
}
```

FIGURE 6.4: YANG model of the configuration data responsible for the GCLs in a TSN switch

6.2.2.2 NETCONF Implementation for TSN devices

The secure transport layer of the NETCONF protocol is not the focus of this work mainly because several protocols such as SSH [197] are already developed for this purpose. Thereby, the implementation of NETCONF in the TSN simulation models starts from the message layer.

A TSN device upon reception of a NETCONF message processes the textual representation of the message using an XML parser and further retrieves the necessary configuration data such as a message type. The message type field, as shown in Figure 6.5 is encoded as the root element of the XML representation and can be set to either notification, RPC or RPC reply. During the configuration process, all TSN

devices except a central configuration entity act as servers and only receive the RPC messages since they need to be configured to offer TSN services.

```
<rpc message-id="1" xmlns="
   urn:ietf:params:xml:ns:netconf:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
  </get-config>
  <filter type="subtree">
    <TSNBridgeConfigurationData xmlns="eti.uni-siegen.
       de:es:yang:tsn">
      <top/>
    </TSNBridgeConfigurationData>
  </filter>
</rpc>
```

FIGURE 6.5: NETCONF get-config RPC queried for configuration data of a TSN device

An RPC, however, can be used to invoke several operations, but this work only supports two main operations: the get-config operation and the edit-config operation. The get-config operation is utilized to obtain capabilities and requirements of TSN devices. More specifically in the TSN simulator, first, the central configuration model that acts as a configuration client sends an RPC message with the get-config RPC to query information about network structure and the active ports of devices. Then, this entity uses the acquired configuration data to compute port-specific GCLs and also to schedule jobs running on the simulated network. Figure 6.5 presents an example of a NETCONF get-config RPC. The TSN simulator only supports the running datastore. Hence as shown in Figure 6.5, the source field of the get-config operation is set to <running/>. On the other hand, a TSN device generates the RPC-reply message while processing an RPC message with the get-config RPC since the RPC and RPC-reply messages share many common attributes such as the message-id. The TSN device after a successful process of an RPC message encodes the requested configuration data under the <data> element of the RPC-reply message and then sends back the message to the central entity. However, if the parsing of the RPC message fails, the device sends back the RPC-reply message with an <rpc-error> element to inform the central entity about errors. Figure 6.6 represents a YANG model which is used to retrieve the configuration information from TSN devices. Additionally, Figure 6.7 depicts the reply message to the NETCONF get-config RPC, which is generated based on the YANG model shown in Figure 6.6. It has to be noted the YANG models used throughout this work are solely defined based on the capabilities and the requirements of the network components in the TSN simulator and most likely cannot directly apply to actual TSN devices.

```
        grouping groupTop {
          leaf name {type string;}
          leaf address {type int64;}
          leaf devicetype {type string;}

          leaf num_ports {type uint16;}
          container ports {
            list port {
              key "port_id";

              leaf port_id {type uint16;}
              leaf connected {type boolean;}
              leaf state {type portstate;}
            }
          }
        }
```

FIGURE 6.6: A simplified YANG model for the configuration data queried by the get-config operations in the TSN simulation framework

```
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1
   .0" message-id="1">
  <data>
    <TSNBridgeConfigurationData xmlns="eti.uni-siegen.
       de:es:yang:tsn">
      <top>
        <name>switch_1</name>
        <address>1111</address>
        <devicetype>switch</devicetype>
        <num_ports>1</num_ports>
        <ports>
          <port>
            <port_id>0</port_id>
            <connected>true</connected>
            <state>3</state>
          </port>
        </ports>
      </top>
    </TSNBridgeConfigurationData>
  </data>
</rpc-reply>
```

FIGURE 6.7: Response to a NETCONF get-config RPC as shown in Fig.6.5

```
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
   message-id="2">
  <edit-config>
    <target>
       <running/>
    </target>
    <config>
       <TSNBridgeConfigurationData xmlns="eti.uni-
          siegen.de:es:yang:tsn">
          <StreamConfiguration>
            <num_TT>1</num_TT>
            <TT>
              <src_address>30</src_address>
              <phase>0.000050</phase>
              <period_duration>0.200000</
                 period_duration>
              <transmission_duration>0.000010</
                 transmission_duration>
              <vlan_id>10</vlan_id>
              <dest_ports>
                <num_dest_ports>1</num_dest_ports>
                <port_id>0</port_id>
              </dest_ports>
            </TT>
          </StreamConfiguration>
       </TSNBridgeConfigurationData>
    </config>
  </edit-config>
</rpc>
```

FIGURE 6.8: NETCONF edit-config RPC containing configuration data to configure stream scheduling in a TSN switch based on the YANG model in Fig.6.3

The TSN simulator also implements the edit-config operation that is used to adjust configuration data of TSN devices. The central configuration model dispatches a separate NETCONF message with an edit-config RPC for each device in the network. The device-specific NETCONF message contains the necessary data which is used to configure the corresponding device. A TSN device creates the stream identity table and transmission schedule tables from the configuration data, including the ingress traffic profiles and the port-specific GCLs. Like the get-config operation, the central entity sets the target field of the NETCONF get-config RPC to the value <running/> since TSN devices in the simulation framework only support the running datastore. Figure 6.8 denotes an example of a NETCONF edit-config RPC where the configuration data is specified based on the YANG model shown in Figure 6.3. Upon reception of a NETCONF message with edit-config RPC, a TSN device first retrieves the configuration data from the content of the message using an XML-parser and then modifies its configuration parameters accordingly. It has to be noted that a TSN device does not always process NETCONF messages successfully.

Therefore, each device replies to the edit-config RPC by sending back a RPC-reply message with either <ok> or <rpc-error> element.

6.2.2.3 Implementation of the CNC/CUC Entity

In the TSN simulator as described before, the central configuration model implements both CNC and CUC entities. The central entity is mainly responsible to acquire knowledge about the device capabilities and the network topology and then based on the collected information computes valid schedules for the jobs running on the network and the associated messages. To this end, the central entity dispatches the NETCONF messages with the get-config RPCs towards all TSN devices. Further, this entity receives the specification of devices in response to the NETCONF get-config RPCs. The acquired information is used for scheduling of jobs and corresponding traffic.

6.2.3 TSN End System Model

In the TSN simulator, the Riverbed regular Ethernet workstation (i.e. ethernet_station_adv) is used as a basis for modelling a TSN-capable end system. Therefore, the Ethernet workstation is modified to accommodate different TSN features, including time-based filtering, clock synchronization, TAS and FRER. The motivation of having a standalone TSN end system drives the integration of TSN services to other existing protocols in this model. Figure 6.9 presents the model of TSN-aware end system in Riverbed.

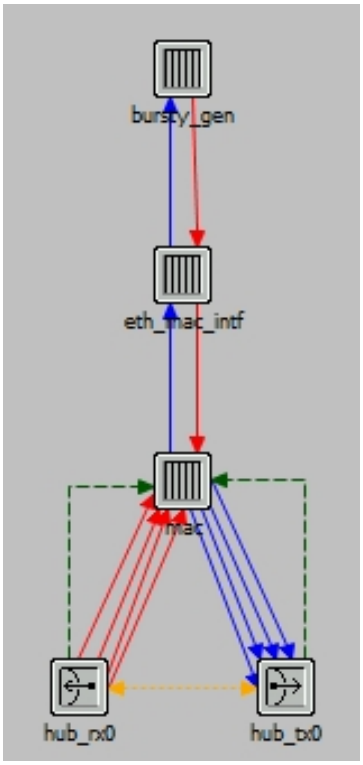


FIGURE 6.9: TSN end system model in Riverbed simulation framework

The end system model comprises four modules as follows:

- local time functionality

- tsn_message_process
- eth_mac_interface
- mac

6.2.3.1 Local Time Functionality

The primary task of a physical clock is to measure time. A clock consists of a counter and a physical oscillation method that triggers an increment of the counter by generating periodic events named the clock microticks. The time interval between two consecutive microticks specifies the granularity of the clock. It can be measured by the nominal number of microticks of the reference clock z during this time interval. Moreover, the frequency ratio between the clock k and the reference clock z at microtick i determines the drift of a clock k. More specifically, the drift as stated in Equation 6.1 is computed by dividing the granularity of clock k with respect to the reference clock z by the nominal number n^k of microticks of the reference clock z [29].

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} \tag{6.1}$$

The drift of a clock can be formulated either linearly or non-linearly. In a clock with linear drift, the drift rate is constant while in a clock with the nonlinear drift, the drift rate varies according to the environmental factors such as temperature.

In Riverbed simulation framework, all devices use the simulation time as a reference time during the execution of different modules. Thereby, each device model in the TSN simulator originally does not have a clock drift. However, every time-aware system in a real-world network has its drift rate. For this reason, the local clock of each time-aware system in the TSN simulator, which is used to time-stamp gPTP messages, is modelled as follows:

$$local\_time = Sync\_time + time\_drift + clock\_tick \tag{6.2}$$

In Equation 6.2, SyncTime is set to the Riverbed simulation time during the network initialization phase and changes to the grandmaster time after completion of the synchronization process. In the TSN simulation framework, the local clock of the grandmaster refers to the simulation time (i.e. $op\_sim\_time()$). As stated in Equation 6.2, the local time of a time-aware system is calculated from the summation of synchronization time, time drift, mean path delay and time duration of a clock tick.

The clock tick is measured as follows:

$$clock\_tick = current\_absolute\_time - previous\_absolute\_time \tag{6.3}$$

Where the $previous\_absolute\_time$ represents the local time of a time-aware system when it is synchronized to the grandmaster clock. Furthermore, the mean path delay is calculated according to the principle that is illustrated in Figure 3.20.

$$time\_drift = drift\_rate * clock\_tick \tag{6.4}$$

Equation 6.4 presents the linear clock drift that is obtained from the multiplication of the clock tick and the constant drift rate of a time-aware system. In the TSN simulator, the value of the drift rate for every time-aware system is assigned by a user.

$$\begin{aligned} compensated\_time &= local\_time - compensated\_time\_offset \\ compensated\_time\_offset &= (NRR + CSRO) * clock\_tick \end{aligned} \tag{6.5}$$

Equation 6.5 denotes how the linear clock drift of a slave system is compensated using the NRR and the CSRO ratios. In the TSN simulation framework, it is assumed that the grandmaster has zero drift rate and its local clock is set to the simulation time (i.e. $op\_sim\_time()$) and is further used to time stamp synchronization messages.

$$NRR = \frac{PresOriginTimestamp - MasterPreviousAbsoluteTime}{PresReceiptTimestamp - SlavePreviousAbsoluteTime} \tag{6.6}$$

As shown in Equation 6.4, the clock drift from the reference time (i.e. the grandmaster time) increases linearly over time and results in an inaccurate local time. As described in Section 3.5.5.2, NRR specifies the frequency ratio between two neighbouring systems and along CSRO aim to compensate the clock drift as stated in Equation 6.5. As denoted in Equation 6.6, Neighbor Rate Ratio (NRR) is obtained from the ratio of a time duration of a neighbour time-aware system and a time duration of a desired time-aware system. In this equation, *PresOriginTimestamp* and *PresReceiptTimestamp* parameters refer to the timestamps which are associated with either sent or received synchronization messages and have the highest drift from the grandmaster time. On the other hand, *MasterPreviousAbsoluteTime* and *SlavePreviousAbsoluteTime* are representatives of the local time with zero drift from the grandmaster time. It is visible when the local clocks of two neighbouring time-aware systems have an identical drift rate, the NRR would be one. Calculating NRR can be seen as a linearization process because NRR is used to compensate the clock drift. The computation of NRR for two time-aware systems with the linear drift is rather simple. On the contrary, for neighbouring time-aware systems with non-linear clock drifts, the frequency ratio is calculated approximately. Therefore, in the latter case, selecting the appropriate synchronization interval affects the accuracy of the synchronization process significantly.

As described earlier, the frequency ratio between two neighbour time-aware systems with identical drift rates is equal to one which may lead to an inaccurate computation of the synchronized time due to an actual frequency offset between the slave clock and the grandmaster clock. Therefore, Cumulative Scaled Rate Offset (CSRO) is calculated and embedded in synchronization messages to address this frequency offset.

The CSRO is obtained from the summation of NRRs as described in Section 3.5.5.2. It is noteworthy that every time-aware relay between the slave and the grandmaster clock calculates the NRR and updates the CSRO field of a synchronization message accordingly. As a result, even the slave time-aware systems with unity NRR can compensate their drift from the grandmaster clock using the CSRO ratio.

In Equation 6.5, CSRO and NRR ratios are used to compensate the slave clock drift in a way that the compensated time converges to the grandmaster time. In the TSN simulator, the compensated time is utilized as a reference time for scheduling and filtering.

6.2.3.2 tsn_message_process

In this model, the tsn_message_process module is responsible for TSN services such as TAS and FRER. This module comprises one main process and five child processes as follows:

- tsn_be_source
- tsn_rc_source
- tsn_tt_source
- tsn_end_station
- tsn_dequeue

The main process model defines the status flag parameters, which are later used to determine whether the associated child process is available for invocation or not. It also spawns the child processes mentioned above. In TSN networks, an end-system can be either talker or listener. Thereby, depending on the role and traffic type, a specific child process is invoked, and the particular flow control mechanism is executed. It has to be noted that the FRER logic of end-system and switch models is identical.

For transmitting TT streams, the **tsn_tt_source process** is invoked. In this process, first, the source module generates a frame according to the outgoing TT traffic profiles. After that, the frame is passed to the sequence generation module to compute the appropriate sequence number. This module stores the last generated sequence number for every TT stream as a parameter called LastGenSeqNum in the stream identity table. For the first frame, the sequence generation function sets the LastGenSeqNum to one. For subsequent frames, this function increments the LastGenSeqNum by one. It is noteworthy that in the TSN simulator the LastGenSeqNum is reset to zero after reaching the value 50. As a next step, the sequence encoding function encodes the stream's LastGenSeqNum into the R-TAG format. Then, the enqueue module puts the frame into the TT queue. The end-system model uses the queuing scheme as illustrated in Figure 6.10.

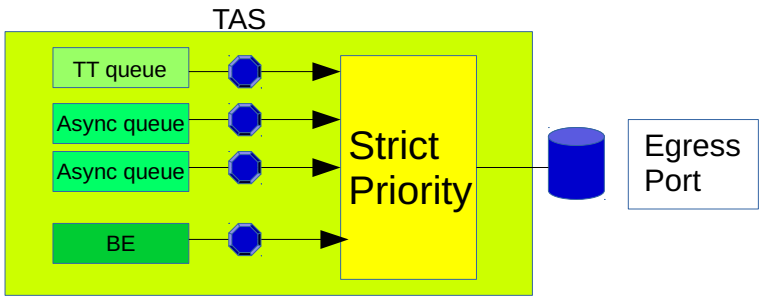


FIGURE 6.10: The queuing scheme of the TSN end system model

The primary task of the **tsn_dequeue process** is to send the generated frames to the lower layer (i.e. MAC layer). To this end, the process continually examines the status of the queue gates through the egress time-aware shaping module. On the other hand, the dequeue module in this process is responsible for sending out the frames from the eligible queue to the MAC layer. For instance, to dequeue TT frames, first, the egress time-aware shaping function checks whether the gate of the

TT queue is enabled according to the given GCL. Then, if the gate of the TT queue is open, the dequeue module sends out the frame over the directly attached link.

The **tsn_rc_source process** is invoked to generate RC frames. As shown in Figure 6.11, the RC flow control mechanism on the transmission side is similar to the TT flow control mechanism. The only difference is related to the dequeue module. In the stream identity table, for every RC stream, the time instant that the last frame of a specific stream was sent is stored in a parameter called LastSentFrame. For each RC frame before dequeuing, the current local time is checked against the stream's LastSentFrame. The RC frame is transmitted only if the time interval between the current local time and the LastSentFrame is equal or more than the specified BAG. Otherwise, the RC frame will remain in the queue.

For dispatching BE traffic, the **tsn_be_source process** is invoked. As denoted in Figure 6.11, the FRER logic is excluded from the BE flow control mechanism since the loss of BE traffic does not cause any critical failures. However, like other types of traffic, the egress time-aware shaping function retrieves the status of the corresponding BE queue's gate. If this function declares the gate of the target queue as enabled, the dequeue module sends out the BE frame over the attached link.

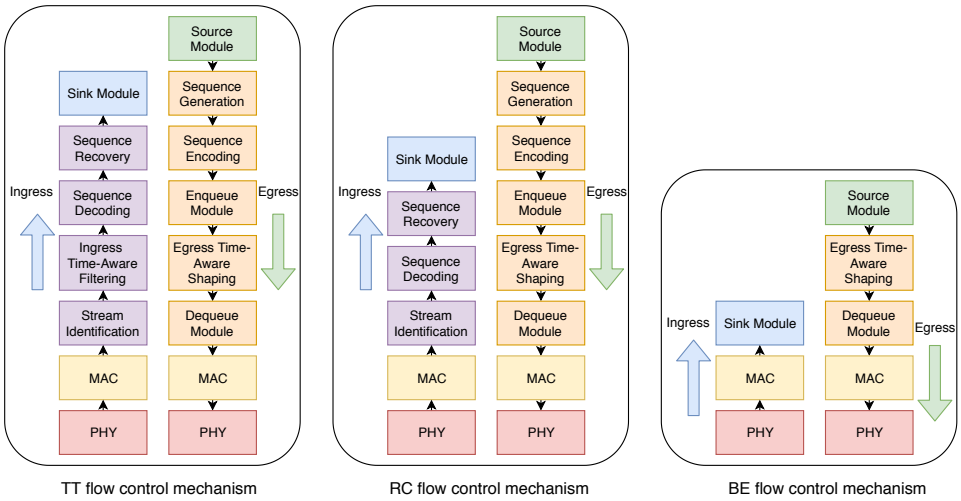


FIGURE 6.11: Ingress and egress flow controls of a TSN end-system

Figure 6.11 presents the ingress and egress flow control mechanisms within a TSN end-system model.

The time-aware end system plays the role of an 802.1ASRev ordinary clock with the specific functionalities. For this reason, the **tsn_end_station process** in a TSN switch model incorporates BMCA, synchronization and peer delay measurement mechanisms.

During the initialization phase, a time-aware system generates gPTP Announce messages to communicate its configuration parameters which are derived from the external clock source and broadcasts the messages to all other time-aware systems residing in a gPTP domain. The state-decision algorithm later uses this configuration information for the establishment of a master-slave hierarchy. Figure 6.12 depicts the structure of the Announce message.

As shown in Figure 6.12, an Announce message contains the following fields:

- currentUtcOffset: Time difference between TAI and UTC with respect to the active grandmaster.

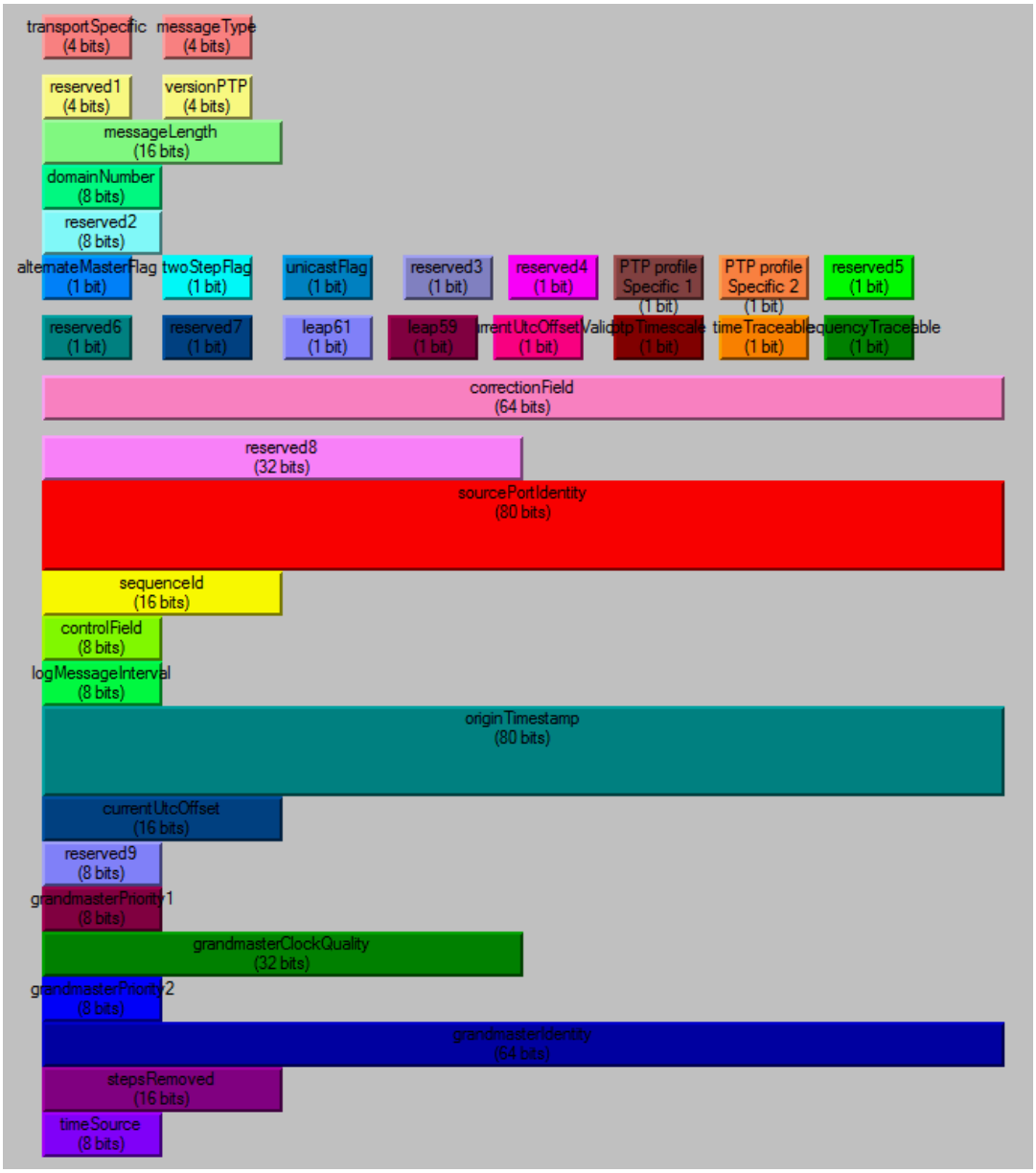


FIGURE 6.12: Announce message format

- grandmasterPriority1: This is a one-octet user-configurable field where the lowest value maps to the highest priority. In a gPTP network, the value of this field for master capable devices is set to 128 while for slave clocks it is set to 255.
- grandmasterClockQuality: The field consists of the Clock Class, Clock Accuracy and Clock Variance.
- grandmasterPriority2: This field like the grandmasterPriority1 is user-configurable and used to identify primary and backup grandmaster clocks among clocks with the same attributes.
- grandmasterIdentity: This consists of port information of a time-aware system.
- stepsRemoved: The field is used by a receiver to verify the correctness of the Announce message.

- timeSource: This specifies the type of time source used by a time-aware system.
- Path trace Type Length Value (TLV): It determines the devices that an Announce message traverses.[17]

In the TSN simulator, every time-aware system uses an internal clock. Besides, only two fields of an Announce message (i.e. Priority1 and Priority2) are used to identify the best master clock concerning the drift rate.

The BMCA is divided into two algorithms: The data set comparison algorithm compares its clock data set with the data sets of other time-aware systems. After that, the state decision algorithm determines the role of a time-aware system and also the state of every port. Further, the algorithm updates the data set of the local clock accordingly. Each time-aware system in the TSN simulator apart from its clock data set, maintains a parent clock data set containing two attributes for the current grandmaster. Before dispatching the first Announce message, the parent dataset is set to the local clock dataset. Upon reception of an Announce message, the tsn_end_station process executes BMCA, where the received clock data set, is compared with the parent data set. The time-aware system is identified as a slave clock and the parent data set is changed to the received clock data set when the clock data set of the sender is better than its own and the parent clock data sets. In contrast, if the clock data set of the recipient time-aware system is better than the sender data set, then the receiver time-aware system is configured as a master clock.

For a better understanding of BCMA execution, consider a gPTP domain with three time-aware end stations where the priorities of system 'A' are 128 and 128, the priority1 and priority2 of system 'B' are 128 and 129 respectively, and the priorities of system 'C' are 128 and 130. It is worthy to note that on start-up all time-aware systems configure their priority attributes and also update the corresponding parameters in the parent data sets. Once all time-aware systems dispatch their first Announce messages, each system would receive two Announce messages from its neighbouring devices. When system 'B' receives the Announce messages originated from system 'A', it first compares its data set with the system 'A' data set and then identifies itself as a slave clock. The reason behind this decision is that the system 'A' has a better data set. Therefore, system 'B' updates its parent data set with the system 'A' dataset. On the contrary, when this system compares its data set with system 'C' data set, it recognizes itself as a better clock source. However, its parent data set (i.e. system 'A') still presents a better clock. As a result, system 'B' remains as a slave clock. Systems 'A' and 'B' also perform the same procedure and at the end, system 'A' is selected as a grandmaster in the described gPTP domain mainly because it has a better clock data set compared to systems 'B' and 'C'.

Upon reception of a Pdelay_Req message, the tsn_end_station process executes the peer delay mechanism. If a TSN end system plays the role of a slave clock, it sends Pdelay_Req messages periodically to neighbour devices. Figure 6.13 presents the Pdelay_Req message structure.

When a time-aware system receives a Pdelay_Req message, first it stores the time instant at which the message is received (i.e. receiving timestamp) and then creates a Pdelay_Resp message containing this value. In the TSN simulator, a time-aware system also appends to a Pdelay_Resp message the sending timestamp of the message mainly because it implements the one-step clock processing. Figure 6.14 illustrates the format of the Pdelay_Resp message.

Once a slave clock receives a Pdelay_Resp message, the time-aware system has all four timestamps required for peer delay measurement. Thereby, it calculates the mean path delay based on Equation 3.10 as follows:

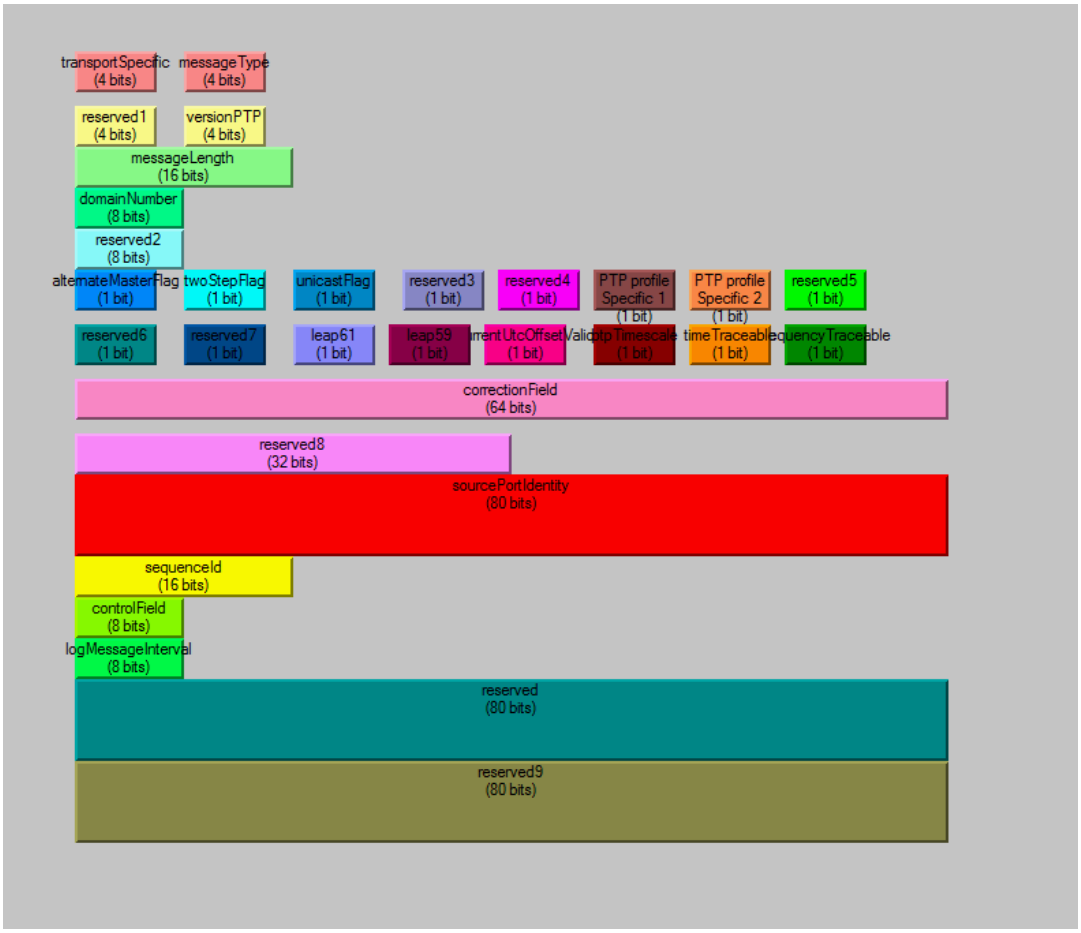


FIGURE 6.13: Pdelay_Req message format

$$mean\_path\_delay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{6.7}$$

Where t_1 is the egress timestamp of the Delay_Req message, t_2 is the ingress timestamp of the Delay_Req message, t_3 is the egress timestamp of the Delay_Resp message and t_4 is the ingress timestamp of the Delay_Resp message. The slave clock maintains the measured mean link delay and later uses it during the synchronization process.

The tsn_end_station process, in addition to BMCA and Peer delay mechanism, implements the synchronization module. This method is executed upon reception of Sync messages to syntonize the local clock to the grandmaster time. To this end, a slave gPTP device measures the time difference between the slave clock and the grandmaster clock based on data embedded in a Sync message as follows:

$$\begin{aligned} \textit{Time difference from grandmaster} = \textit{ingress timestamp Sync} - \textit{egress timestamp Sync} - \\ \textit{link delay between slave and grandmaster} - \textit{correctionField of Sync} \end{aligned} \tag{6.8}$$

Where *ingress timestamp Sync* indicates the time instant at which the Sync message is received, *egress timestamp Sync* presents the time instant at which the grandmaster clock sends Sync message, and *correctionField of Sync* contains the residence time of a Sync message in intermittent devices.

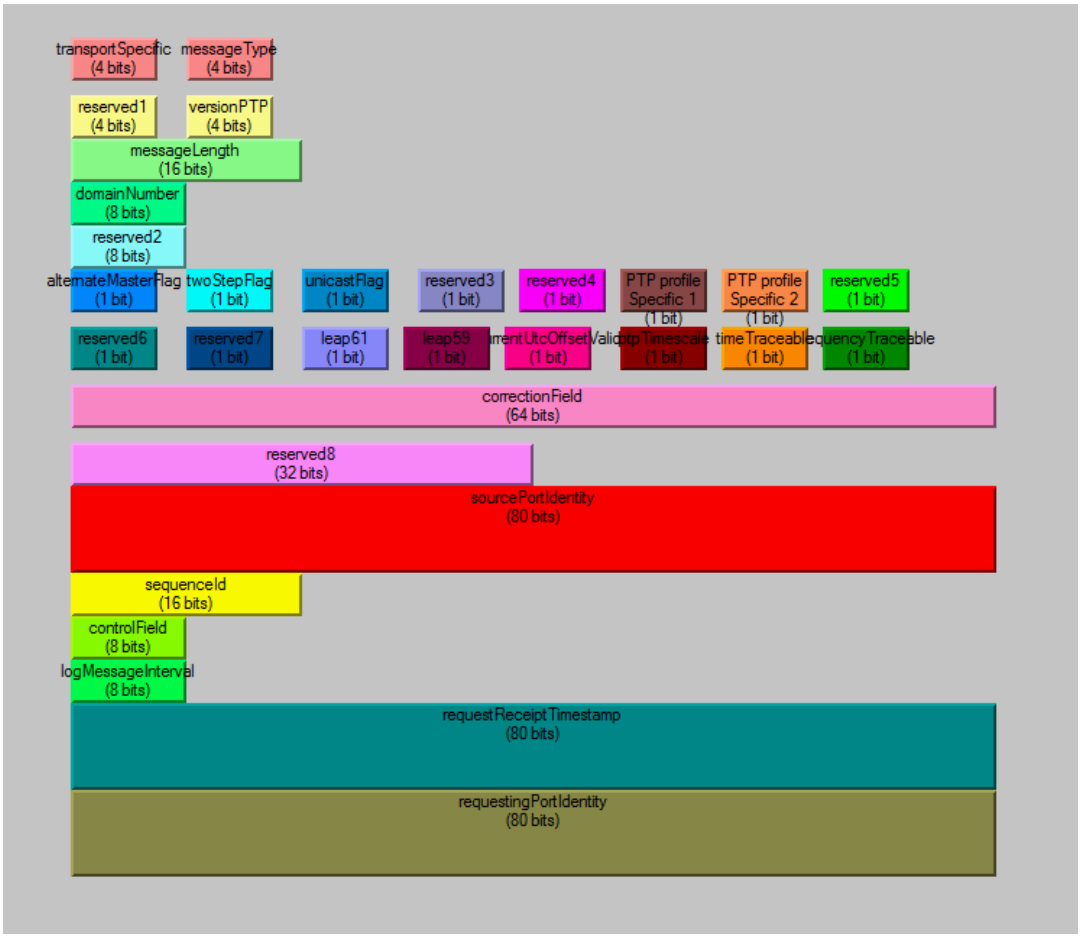


FIGURE 6.14: Pdelay_Resp message format

A time-aware system identified as a grandmaster clock sends the Announce and Sync messages to slave clocks periodically. The grandmaster clock also sends back a Pdelay_Resp message in response to a Pdelay_Req message originated from a slave gPTP device. However, a slave gPTP device schedules an announce_receipt_timeout event to examine the state of a current grandmaster clock periodically. If a slave clock does not receive any Announce message from the active grandmaster clock within announce_receipt_timeout interval, it starts sending an Announce message to other time-aware systems to select a new grandmaster clock. It is good to mention that in the tsn_end_station process, Sync and Announce messages are scheduled at different cycles. Figure 6.15 represents a Sync message format.

6.2.3.3 eth_mac_interface

The TSN end system model uses the eth_mac_interface process of the standard Ethernet workstation without any modification. This process sets up an Interface Control Information (ICI) structure which is used to exchange data between the higher layer processes (e.g. tsn_message_process) and the mac process.

6.2.3.4 MAC Process

The mac process receives frames from either the higher layer or the physical layer. For a frame that is received from the physical link, the mac process first invokes the

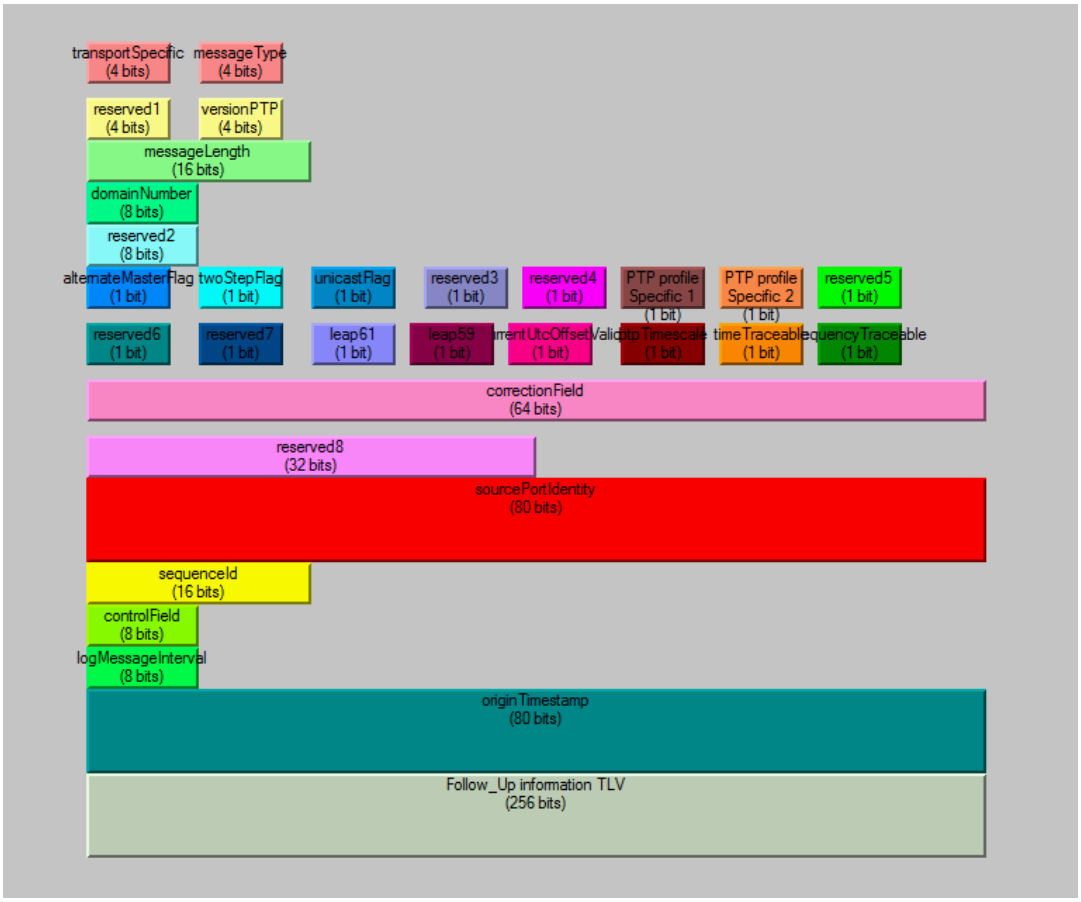


FIGURE 6.15: Sync message format

stream identification function to determine the stream that the frame belongs to using the source MAC address and the VLAN ID fields. Depending on the traffic type, the mac process uses different flow control mechanisms. For TT streams, the ingress time-based filtering function checks the arrival time of a TT frame against the current local time. If the frame belongs to a TT flow that arrives outside the predefined reception window, the filtering function will drop the frame. In the counter case where a TT frame arrives at the expected time interval, the function passes control to the sequence decoding function. The sequence decoding function retrieves the sequence number of the frame from the last two octets of the R-TAG header. In the stream identity table, the last received sequence number of every RC and TT stream is stored in a parameter called LastRecSeqNum. Therefore, the sequence recovery function compares the sequence number of the frame against the stream's LastRecSeqNum. If the frame carries the sequence number other than LastRecSeqNum + 1, the sequence recovery function discards the frame. Otherwise, it passes control to the sink module mainly because this function is designed based on the VectorRecoveryAlgorithm [28]. In the end, the sink module terminates the receiving frames.

As shown in Figure 6.11, on the reception side, the flow control of RC streams is similar to the TT flow control. The only difference is that the ingress time-based filtering function would not be invoked for this type of traffic. Moreover, the received BE traffic from the physical layer is terminated at the sink module without any further processing.

The timestamping operation plays a crucial role in a time-aware system mainly because it determines the synchronization accuracy, which is provided by the IEEE

802.1As-Rev protocol. A gPTP device could achieve a synchronization accuracy as good as one microsecond if timestamping takes place close to the physical layer. In an actual time-aware system as illustrated in Figure 6.16, the timestamp operation occurs at the sublayer MII which is located between the MAC and Logical Link Control (LLC) sublayers within the Data Link Layer (DLL). However, a TSN end system model in Riverbed simulator does not have the MII layer. Consequently, the timestamping of event message happens either when the message is sent from the mac module towards the transmitter module (hub_tx0) or when it is sent from the receiver module (hub_rx0) towards the mac module.

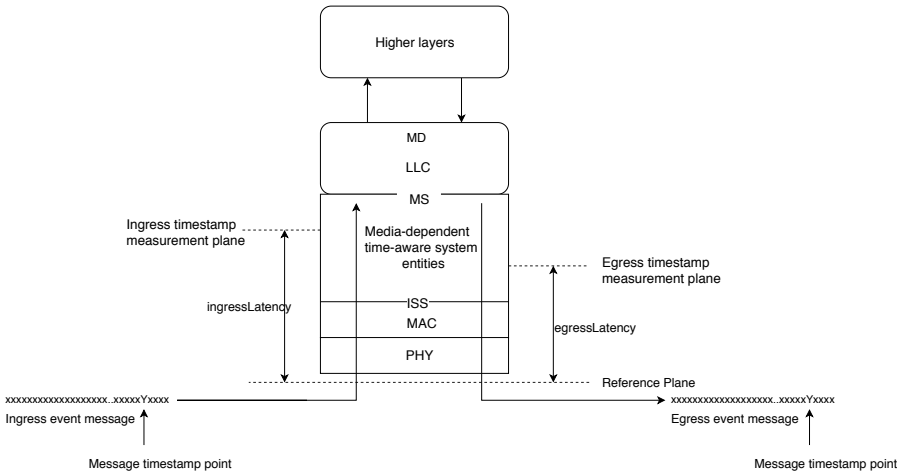


FIGURE 6.16: gPTP timestamping mechanism [17]

As shown in Figure 6.16, the timestamping of an event message within a time-aware system takes place when the message timestamp point passes the reference plane which separates the network from a gPTP device. The timestamps are generally created at a timestamp measurement plane which results in an additional delay between the reference plane and the measurement plane. For this reason, the introduced time offsets are removed from the generated timestamps of either incoming or outgoing event messages, as shown in Equation 6.9.

$$\begin{aligned} egress\_timestamp &= egress\_measured\_timestamp + egress\_latency \\ ingress\_timestamp &= ingress\_measured\_timestamp - ingress\_latency \end{aligned} \tag{6.9}$$

In Equation 6.9, the timestamps relative to the reference plane (i.e. *egress_timestamp* and *ingress_timestamp*) are calculated based on the timestamps relative to the measurement plane (i.e. *egress_measured_timestamp* and *ingress_measured_timestamp*) and the corresponding delays (i.e. *egress_latency* and *ingress_latency*). Since the timestamping of an event message should occur when the Start of Frame Delimiter (SFD) of the message is passed to the reference plane, the equation mentioned above can be formulated more precisely as follows:

$$\begin{aligned} actual\_egress\_timestamp &= egress\_timestamp + \frac{(preamble+SDF)}{data\ rate} \\ actual\_ingress\_timestamp &= ingress\_timestamp - \frac{packet\ size - (preamble+SDF)}{data\ rate} \end{aligned} \tag{6.10}$$

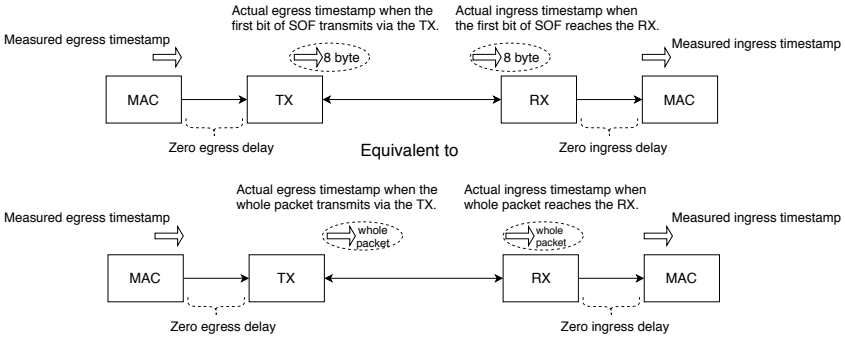


FIGURE 6.17: Timestamping point in TSN end system model [66]

Figure 6.17 depicts how the timestamping operation is implemented within a TSN end system model in Riverbed. In the TSN simulation framework, *egress_latency* and *egress_latency* are zero since the mac module is attached to the transmitter and the receiver modules directly via point-to-point connections (See Figure 6.9). Furthermore, the transmission delay between a time-aware system and the network is negligible (in the order of nanoseconds) mainly because the 100 Gbps physical links are used to interconnect gPTP devices. Thereby, in the TSN end system model *egress_timestamp* and *ingress_timestamp* are expressed as:

$$\begin{aligned} egress\_timestamp &= egress\_measured\_timestamp \\ ingress\_timestamp &= ingress\_measured\_timestamp \end{aligned} \tag{6.11}$$

When a grandmaster clock sends Sync messages towards slave clocks, the egress timestamp relative to the local clock is embedded into the originTimestamp field of a Sync message. Moreover, when a time-aware system receives a Pdelay_Req message, it records the ingress timestamp of a message and stores the timestamp into the receiveTimestamp field of the subsequent Pdelay_Resp message. On the other hand, a slave clock retrieves the timestamp encapsulated in a Sync message during the synchronization phase. Besides, it stores the egress timestamp of the Pdelay_Req message and uses it for the link delay measurement.

6.2.4 TSN Switch Model

For modelling a TSN switch, the Riverbed Ethernet switch model called "ethernet16_switch_adv" is modified in a way that it supports subsets of non-time-based and time-based features of TSN.

The regular Ethernet switch forwards the incoming frames to the target device based on the MAC destination address since it operates at link layer of OSI. If the destination address of the frame is set to an Ethernet broadcast address, the switch sends out the copies of the frame from all egress ports. Aside from broadcast frames, the Ethernet switch supports multicast traffic patterns. Furthermore, the switch participates in identifying the spanning tree topology through the Bridge Protocol Data Unit (BPDU) messages.

The TSN switch model comprises three different modules, namely local time functionality, switching and MAC.

6.2.4.1 Local Time Functionality

The local time functionality of a TSN switch model and a TSN end system model is implemented identically. Therefore, a TSN switch in a similar way to an end system model contains time attributes such as local time and timestamps related to peer delay measurements and further uses these values to calculate the grandmaster time.

6.2.4.2 Switch Module

The switch module operates through one main process and seven child processes. The main process first sets up all egress ports based on the device attributes. Then, it creates a port map table where the status of every port is stored. Moreover, the main process spawns seven child processes. Each child process is invoked by the main process to handle a specific type of incoming traffic. The following sections describe the mentioned child processes briefly.

bridge_protocol_entity This process runs the spanning tree algorithm upon reception of BPDU messages.

lacp_l2 This process executes the Link Aggregation Control Protocol (LACP) for every enabled port to identify the state and the operational mode of the port. Once the execution of LACP is completed, the lacp_l2 process stores the LACP related information of each port.

mst_bpe This process runs the Multiple Spanning Tree Protocol (MSTP) upon reception of BPDU messages. After that, it stores the spanning tree data associated with each tree instance.

pvst_bpe This process like bridge_protocol_entity and mst_bpe processes runs STP upon reception of BPDU messages. However, it stores data associated with the IEEE 802.1D protocol.

TT_bridge_mac_relay The main process spawns a TT_bridge_mac_relay and an RC_bridge_mac_relay process for every port since the destination ports of TT and RC streams are defined in the corresponding traffic profiles. On the contrary, all BE messages are handled in a BE_bridge_mac_relay process regardless of the outgoing port.

In the TSN switch model, all frames received from the mac module are passed to the main process. This process calls the stream identification function for all incoming frames. As described in Section 3.7.4.3 FRER introduces different stream identification functions. In the TSN switch model, the main process identifies incoming frames using the source MAC address and VLAN ID fields. Therefore, the stream identification function is passive and does not modify the frame passed either up or down in the protocol stack. The stream identification function using the stream identity table categorizes packets to the following traffic types:

- TT frames
- RC messages
- BE traffic

After identifying a frame as a TT traffic, the main process invokes the TT_bridge-_mac_relay process corresponding to the destination port of the frame. As a next step, the TT_bridge_mac_relay process executes the **ingress time-based filtering function**. This function plays a critical role in TSN switches since the timing requirement of the TT communication is strict and vital. The ingress time-aware filtering function checks the arrival time of a TT frame against the current local time of the switch model. If a TT frame arrives outside its reception window, the filtering function will drop the TT frame. Consequently, this function protects the switch from the faulty devices that are trying to delay TT frames or flood the network with unexpected TT flows. In the opposite case, if the frame belongs to a TT flow that is expected to arrive at the current local time, the function passes control to the sequence decoding module.

The **sequence decoding function** determines whether the packet contains an R-TAG header or not. If the frame carries the R-TAG header, the function retrieves the sequence number from the last two octets of the R-TAG header and then calls the sequence recovery function. On the other hand, if the frame does not contain an R-TAG header which means the frame is sent from a FRER-unaware device, the sequence generation function is invoked. This assumption provides an opportunity to simulate a TSN network with a combination of FRER-unaware and FRER-aware network components.

The stream identity table, which is created during the configuration phase, maintains the last received sequence number of each RC and TT stream in a parameter called LastRecSeqNum. The **sequence recovery function** examines the sequence number of the received frame against the LastRecSeqNum of the stream to which the frame belongs to. This function is designed based on the VectorRecoveryAlgorithm. Therefore, if the frame's sequence number is equal to the expected sequence number (i.e. LastRecSeqNum + 1), the enqueue module is invoked. Otherwise, the frame with the wrong sequence number would be dropped.

For the first frame of a specific stream, the **sequence generation function** sets the LastGenSeqNum to zero. For subsequent frames, this function increments the LastGenSeqNum by one. As a next step, the sequence encode function is called. It is noteworthy that the LastGenSeqNum is reset after reaching the value 50.

The **sequence encode function** encodes the LastGenSeqNum of the stream which is set by the sequence generation module in the R-TAG format. To be more specific, the function creates the R-TAG header for the frame and sets the last two octets of the R-TAG to the value of the LastGenSeqNum parameter. After encoding the sequence number, control is passed to the enqueue module.

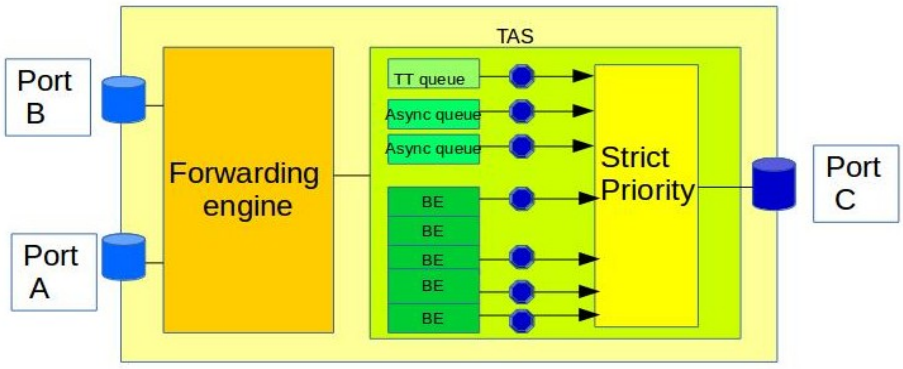


FIGURE 6.18: Queuing scheme of TSN switch model

The time-aware shaping mechanism is only applied to egress port queues. The switch model can use the egress queues for shaping traffic only when the QoS-enabled attribute is set. For this reason, in the TSN switch model the QoS parameter is enabled. The enqueue module puts the frame to the appropriate egress queue according to the priority of the frame. The queuing scheme of a TSN switch model is presented in Figure 6.18. It has to be noted that in the standard Ethernet switch model, the internal priority of a frame is derived from the information encoded in the packet header (e.g. PCP in the Ethernet header or Differentiated Services Code Point (DSCP) in the IPv4 header) and is used to map packets to the corresponding egress queues. However, the TSN switch model does not follow this principle for the TT frame. Instead, it assigns the internal priority based on the traffic type to reflect the priority of TT streams over the other traffic classes. For non-TT traffic, the frame's internal priority is directly derived from the data encapsulated in the message header. In the TSN simulation framework, it is assumed that the highest priority is reserved for the safety-critical traffic (i.e. TT flows). This implies that the priority of non-TT frames cannot be set to the highest priority in the simulated TSN network.

In the TT_bridge_mac_relay process, the **enqueue module** places all TT messages destined to a specific port to the TT queue associated with that port. After enqueuing the frame, the control is passed to the egress time-aware shaping function. The egress time-based shaping function specifies which queue is eligible to transmit the next packet. If the TT queue's gate is open according to the port's GCL, this function checks the required time for the frame transmission against the time interval where the state of the gate remains enabled. If the time interval is sufficient for transmission of the entire frame, the dequeue module is invoked. Otherwise, the frame remains in the queue. This check mainly is performed to prevent initiating a non-TT frame transmission in its time slot and continuing it over a scheduled time slot. To guarantee deterministic behaviour, the TSN switch reserves a fixed time slot called guard band before each TT time slot. The guard band is usually set to the required time for forwarding the Ethernet frame with the maximum length (i.e. 1526 bytes). This approach results in a reduction of the bandwidth usage. Therefore, the presented TSN switch model uses a dynamic guard band, in which the required time for the transmission of a non-TT message is checked against the GCL, instead of a static one. This design decision significantly improves link bandwidth utilization.

It is quite likely that the gate of more than one queue is enabled. In such cases, the transmission selection module decides which frame is sent out over the physical link. This selection is made based on the switch's queuing policy. In this TSN switch model, as shown in Figure 6.18, each egress port has eight queues, and a strict priority selection scheme is applied among TT, RC and BE queues. At the final stage, the **dequeue module** transmits the packet from the TT queue to the attached link.

RC_bridge_mac_relay In the switch module, the main process upon reception of RC frames invokes the RC_bridge_mac_relay process corresponding to the destination port of the frame. Moreover, the stream identity table for every RC stream maintains the time instant that the last frame belonging to the stream was sent in a parameter called LastSentFrame. As denoted in Figure 6.19, the RC_bridge_mac_relay process handles the incoming RC frames in a similar way to the TT_bridge_mac_relay process. However, the ingress time-base filtering function in this process checks the arrival time of an RC frame (i.e. current local time) against the corresponding RC stream's LastSentFrame parameter which is different from the ingress time-based filtering functionality of TT flows. For each RC frame before dequeuing, the current

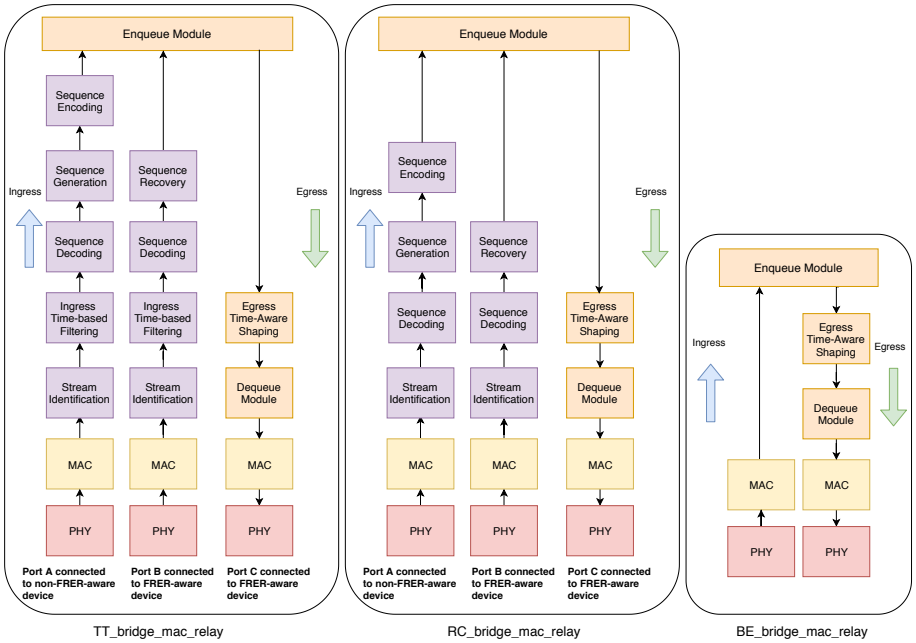


FIGURE 6.19: Packet processing phases in the TSN switch model

local time is checked against the stream's LastSentFrame. The RC frame is passed to the enqueue module only if the time interval between the current local time and the LastSentFrame is equal or more than the stream's BAG. Otherwise, the RC frame is dropped.

An additional difference is related to the enqueue module where RC packets are put into either queue number 6 or queue number 5 depending on RC frame's priority which is defined in the RC traffic profile (priority one maps to queue six and priority zero maps to queue five). It is noteworthy that TT and RC frames can be passed to more than one TT_bridge_mac_relay and RC_bridge_mac_relay processes. For this reason, the switch model does not require a separate split stream function for FRER mechanism.

BE_bridge_mac_relay Upon reception of a BE message, the main process in the switch module invokes the BE_bridge_mac_relay process. As shown in Figure 6.19, the BE_bridge_mac_relay process does not contain the FRER logic mainly because the loss of BE traffic does not result in any severe failures. In the BE_bridge_mac_relay process, the enqueue module specifies the destination port of BE traffic using the MAC database and stores the message in an appropriate BE queue depending on the frame's priority. Like TT flows and RC streams, the BE frame is sent out when the gate of the corresponding queue is enabled for a period of time that is sufficient for transmitting the entire frame.

tsn_bridge_mac_relay_entity The time-aware switch plays the role of an 802.1AS-Rev boundary clock with the respective functionalities. To this end, the tsn_bridge-_mac_relay_entity process implements all TSN clock synchronization methods including BCMA, link delay measurement and synchronization mechanisms. The logic of all TSN clock synchronization mechanisms in a TSN switch model is the

same as the corresponding functionalities in a TSN end system model. In more details, if a time-aware switch is identified as a grandmaster clock, the tsn_bridge_mac_relay_entity process dispatches Announce and Sync messages periodically. In contrast, if a time-aware switch is marked as a slave clock, the process upon reception of Sync and peer delay messages executes the synchronization and peer delay measurement methods respectively.

6.2.4.3 MAC Module

The mac module of a TSN switch model operates in the same way as the mac module of an end system model. This means the mac module stores the timestamps corresponding to all incoming gPTP messages (e.g. Pdelay Req and Pdelay Resp reception time). On the other hand, this module timestamps all outgoing gPTP messages (e.g. Pdelay Req and Pdelay Resp transmission time). Moreover, the mac module minimizes the potential synchronization error by running the synchronization procedure instantly after the reception of a Sync message.

When a TSN switch is marked as a grandmaster clock, all its ports go to the master state and start sending Announce and Sync messages to all slave clocks within the same gPTP domain. Furthermore, a TSN switch with grandmaster role does not perform the link delay measurement since it does not require to run the synchronization procedure. In the counter case, when a TSN switch is marked as a slave clock, at most one port goes to the slave state and the rest of the ports transit to the master state. A mac module corresponding to a port in the slave state, stores all timestamps related to peer delay messages. These values are later used to measure the link delay.

6.2.5 Fault Injector

FRER is introduced to protect the TSN systems against faulty behaviours. To validate safety and fault tolerance mechanisms that are offered by FRER, a fault injector model is developed in the TSN simulation framework. This model simulates different faults (e.g. link failure). Therefore, the fault injector provides an opportunity to investigate the behaviours of TSN capable devices in such fault states. The fault injector emulates the following faults:

- Link Failure: This failure can be emulated by enabling the link failure parameter for a specific link.
- Crash Failure: The crash failure can be emulated by setting the device-specific failure/recovery attribute.
- Stuck Transmitter: This failure occurs when a sender transmits messages with the same sequence numbers instead of incremental values. To simulate this failure, the end-system model sends frames with repetitive sequence numbers.
- Omission: The omission failure happens when a sender fails to transmit a particular frame, or a specific packet is not delivered to the receiver. The sending end-system emulates this failure by sending frames with non-consecutive sequence numbers.
- Resequencing: When the frames which belong to a particular stream arrive at the receiver in a wrong order, the resequencing failure occurs. To simulate this failure, the sending end-system delays transmitting frames with the lower

sequence numbers. Hence, it first sends frames with the higher sequence numbers.

6.3 Experiments and Evaluation

The main goal of the TSN simulator is to emulate a multi-hop switched Ethernet network in which all devices support time-based (e.g. IEEE 802.1Qbv and 802.1Qci) and non-time-based features (e.g. IEEE 802.1CB) of TSN and communicate with each other via full-duplex 100 Gbps physical links. This simulation executes on a PC with 32 GB memory and a dual-core 2.4 GHz CPU.

6.3.1 Experimental Setup

To evaluate the TSN simulation framework with respect to determinism, fault-tolerance, configurability and clock synchronization, an example layout of an Ethernet-based Train Communication Network (TCN) [141] is used because it provides a realistic setup for studying different aspects of TSN features. This network, like other Ethernet-based train topologies comprises an Ethernet Train Backbone (ETB) and an Ethernet Consist Network (ECN). As shown in Figure 6.20, all devices in ECNs are connected to the ETB via two redundant ETB lines and also two separate ETB switches. Besides, within every consist network, ECN switches are connected using a ring topology. Consequently, this layout offers redundant paths at both ECN and ETB level to meet the high safety demands of mission-critical train applications (e.g. braking system).

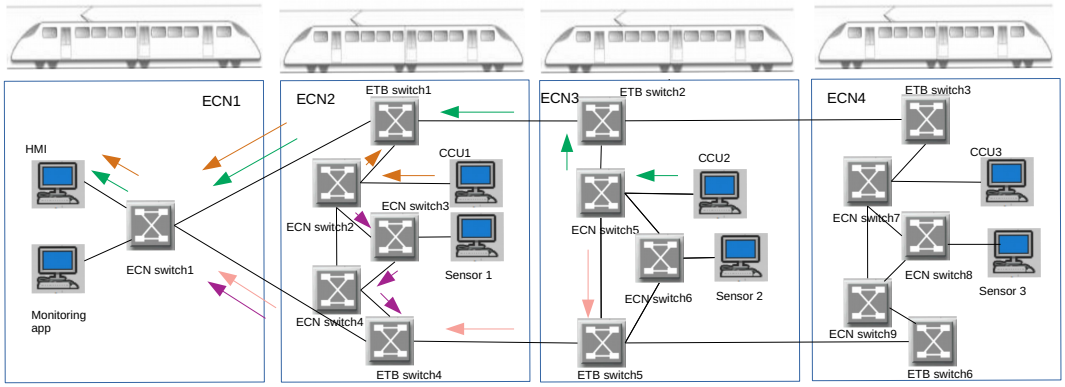


FIGURE 6.20: Experimental network structure. The redundant paths of stream $s1$ are shown by orange and purple arrows while green and pink arrows illustrate the stream $s2$ redundant routes.

In this setup, within every consist network (except ECN1), each sensor first collects data every 100 ms and then sends samples to the corresponding Central Computing Unit (CCU). After that, every CCU sends sensor data to the Human Machine Interface (HMI) and the monitoring application which resides in ECN1. Finally, the HMI processes the sensor samples and sends back process messages towards the respective CCU. It is noteworthy that all streams mentioned above are assumed as TT traffic. To achieve a more realistic experimental setup, it is assumed that there is background traffic from the monitoring application towards all CCUs. Table 6.1 details the parameters of all TT streams which are sent over the experimental network.

Stream	*Src* → *Dst*	Period (ms)
$s1$	$CCU1 \rightarrow HMI$	200
$s2$	$CCU2 \rightarrow HMI$	200
$s3$	$CCU3 \rightarrow HMI$	200
$s4$	*temp sensor* → $CCU1$	100
$s5$	*light sensor* → $CCU2$	100
$s6$	*door sensor* → $CCU3$	100
$s7$	$CCU1 \rightarrow$ *monitoring app*	200
$s8$	$CCU2 \rightarrow$ *monitoring app*	200
$s9$	$CCU3 \rightarrow$ *monitoring app*	200
$s10$	$HMI \rightarrow CCU1$	200
$s11$	$HMI \rightarrow CCU2$	200
$s12$	$HMI \rightarrow CCU3$	200

TABLE 6.1: TT Stream Specifications

In the experimental network apart from the ECN switches, ETB nodes, sensors and CCUs, there is a configuration manager node that models a CNC/CUC entity. Additionally, in TSN simulator, the NETCONF messages that facilitate the configuration of TSN capable devices, are identified as BE traffic. Consequently, these messages can not be sent over the network before configuring a forwarding spanning tree.

At the beginning of the simulation, RSTP calculates a spanning tree for the experimental network. The convergence of RSTP for the simulated network takes around 5.2 seconds. After that, the configuration manager starts sending NETCONF messages with get-config RPCs toward all devices residing in the network. In response, TSN capable devices send back the RPC reply messages containing their data to the configuration manager within the NETCONF sessions. Therefore, the configuration manager acquires the knowledge on the structure of the entire network and the associated traffic profiles through the RPC replies and further using this data generates the necessary configuration information (e.g. port-specific GCLs) for deploying different TSN features to devices. As a next step, the configuration manager creates a NETCONF message with edit-config RPC per device in a way that each message contains the generated configuration information for the particular device. The initialization phase is concluded once all devices adjust their configuration parameters according to information embedded into the edit-config RPCs. The completion of the configuration process can be observed from the simulation statistics which illustrate the number of sent and received TT frames belonging to the streams listed in Table 6.1 at each device. In more details, the simulation results show that the sensors start sending sample data at $6th$ seconds, mainly because the calculation of the spanning tree takes quite long. The timely start-up process reflects the design decision made regarding the transmission of NETCONF messages based on the best effort paradigm. For instance, this initialization delay meets the requirements of the inauguration process in trains. However, the mentioned startup interval does not satisfy the initialization requirements of several mission-critical systems.

The start-up duration can be shortened significantly if NETCONF messages are configured as TT traffic. To this end, all devices within the network need to be configured statically to exchange NETCONF messages immediately after the simulation startup. In other words, all configuration parameters related to NETCONF messages need to be deployed to devices at the beginning of the simulation and before sending any message. In this case, the configuration process of the simulated network can not be dynamic as the configuration models of IEEE 802.1Qcc are designed to be.

6.3.2 Experiments and Results

The first part of experiments is dedicated to studying the impact of FRER functionalities on the reliability of mission-critical applications. Namely, the behaviours of TSN capable devices in the presence of different transient and permanent faults are investigated by simulating the following test scenarios:

6.3.2.1 Messages with Identical R-TAG Header

In the first scenario, the fault injector forces the HMI to send process messages with repetitive sequence numbers towards the CCUs for specific time intervals (i.e. from 7.2 to 9 seconds). As the graph in Figure 6.21 depicts, CCU1 is not receiving any process message from the HMI during this period. The FRER logic of ECN switch one discards all process messages with the same R-TAG header and does not permit the malformed packets to consume the network resources (e.g. bandwidth and memory). Therefore, FRER protects the TSN network against this transient fault. CCU1 resumes receiving the process messages just after the HMI recovers from this failure (at 9*th* seconds). As the behaviours of all CCUs in the described condition is identical, we only present the results for CCU1.

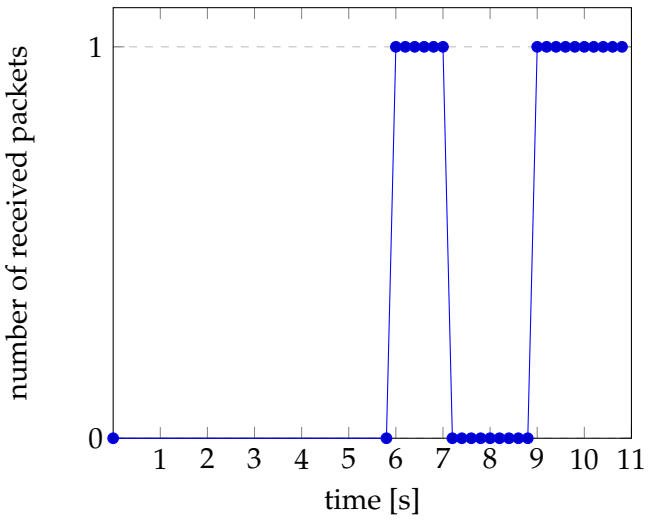


FIGURE 6.21: The number of process packets are received by CCU1 in presence of repetition failure in the HMI

6.3.2.2 Injecting Frames with Wrong Sequence Numbers

To emulate the omission failure, the fault injector modifies the sequence number generation function of CCU1 so that it increments the LastGenSeqNum by three

instead of 1. Consequently, ECN switch 2 notices that some CCU1 messages are missing. Therefore, it discards all frames that originated from CCU1 to mitigate this faulty behaviour. As the graphs in Figure 6.22 denote, the HMI and the monitoring application are not receiving any message from CCU1 just after the fault injected to the network (i.e. at 7.2*th* seconds). CCU1 recovers from this failure at 9*th* seconds and starts to send frames with the consecutive sequence number. ECN switch 2 continues discarding messages that are sent from CCU1, because in ECN switch 2, the LastRecSeqNum of CCU1 stream is not aligned with the R-TAG header of messages sent from CCU1. ECN switch2 resumes accepting CCU1 frames at 13.2*th* seconds. The reason is that the CCU1 messages start carrying the expected sequence numbers due to resetting the LastGenSeqNum parameter in CCU1. The behaviours of the HMI and the monitoring application in this test scenario, as shown in Figure 6.22 are identical.

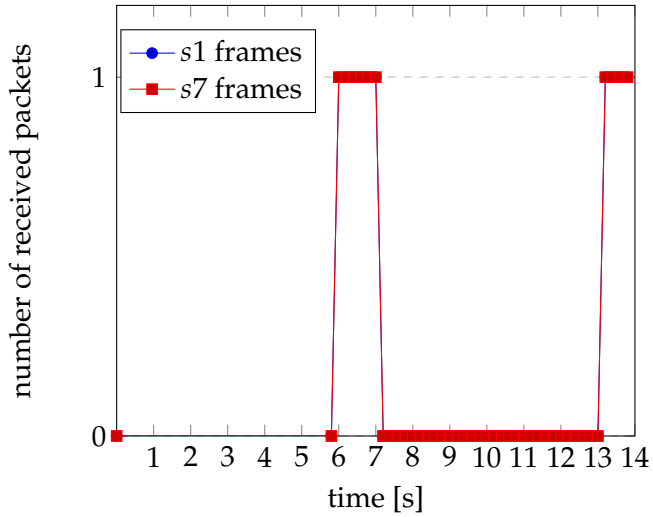


FIGURE 6.22: The number of $s1$ and $s7$ packets received by the HMI and monitoring applications in case of an omission failure

The fault injector also modifies the sequence number generation function of CCU2 so that for every two consecutive frames first it increments the LastGenSeqNum by two and then decrements this parameter by 1. Due to this modification which simulates a resequencing failure, ECN switch five first receives a CCU2 frame with a higher sequence number and then a message with the lower sequence number. Hence, as shown in Figure 6.23, ECN switch five does not forward any CCU2 message to the HMI and the monitoring application just after noticing this faulty behaviour (i.e. at 7.2*th* seconds). In both scenarios as mentioned above, the first switch on the path to the HMI (i.e. ECN switch two and ECN switch five respectively), discards faulty frames until it receives again the frames carrying the expected sequence numbers. Therefore, FRER, in addition to enhancing the network fault tolerance, improves the overall network resource utilization. To achieve this, FRER eliminates the probability of forwarding the malformed packets over the TSN network.

6.3.2.3 Testing FRER against Link Failures

As described before, FRER offers safety and fault-tolerance capabilities using redundant paths. For this purpose, the redundant paths for $s1$ and $s2$ are configured during simulation initialization. It has to be noted in the simulated network TSN

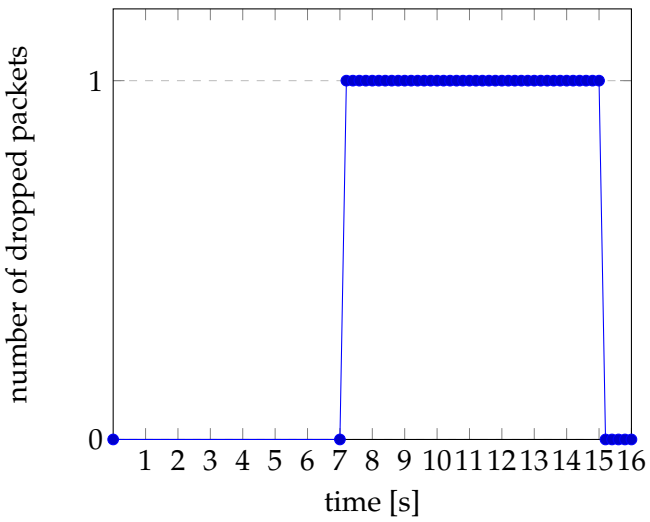


FIGURE 6.23: The number of $s2$ and $s8$ packets that are discarded by ECN switch 5 in case of a resequencing failure in CCU2

capable devices forward the frames belong to $s1$ and $s2$ flows over redundant routes regardless of the states of their ports which are assigned by RSTP.

The fault injector at 7.2th second makes link $l1$ fail. Before the occurrence of the $l1$ failure, ECN switch 1 receives $s1$ and $s2$ frames from two separate paths (which are illustrated in Figure 6.20). Then it forwards the frames which arrive first and eliminates the duplicated messages. As the graph in Figure 6.24a shows, the HMI and the monitoring application do not experience any traffic loss from $s1$ and $s2$. After the $l1$ failure, ECN switch one still receives $s1$ and $s2$ frames from the redundant paths and delivers them to the HMI and monitoring application. These applications are not receiving any $s3$ messages after the $l1$ breakdown. The reason is that no redundant paths are set up for the $s3$ stream. Hence as shown in Figure 6.24a, after the failure in the primary route, $s3$ frames are not delivered to the HMI and monitoring application anymore.

It is good to note that the redundant routes of $s1$ comprise different numbers of links. Consequently, as the graph in Figure 6.24b presents, the end-to-end delay of $s1$ has different values before and after the $l1$ failure. However, the $s2$ end-to-end latency remains unchanged during the simulation. The redundant paths of $s2$ unlike the $s1$ routes have the same number of physical links. In the TSN simulator, every frame remains in the TSN switch for 2 μs (i.e. $D_{\text{processing}}$). Moreover, the size (s_m) of all frames which are sent over our network is set to 64 bytes and the bandwidth (b_l) of all links is 100 $Gbps$. Thereby, the transmission delay (D_t) of a frame leaving the TSN switch is calculated as follows:

$$D_t = s_m / b_l = (64bytes) / (100Gbps) = 5.12ns$$

As the propagation delay ($D_{\text{propagation}}$) of each link is set to 8 μs, the frame's end-to-end delay considering the chosen route is calculated as follows:

$$D_{\text{e2e}} = num.hops * (D_{\text{processing}} + D_t + D_{\text{propagation}})$$

For instance, D_{e2e} of $s1$ before and after the $l1$ failure is computed as follows:

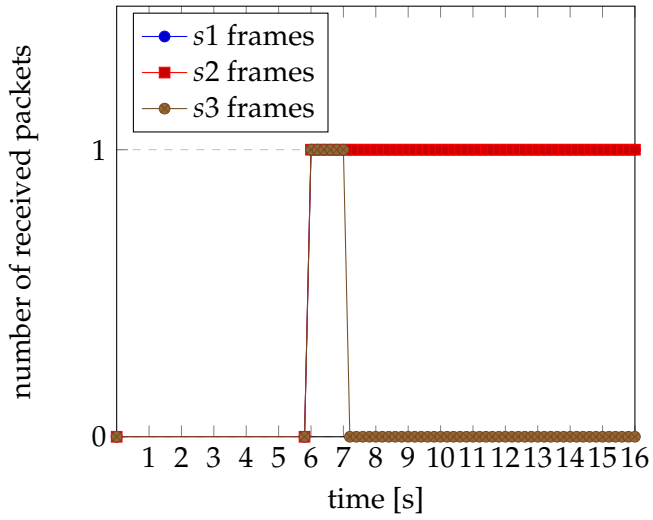
$$\text{Before failure} : D_{\text{e2e}} = 4 * (2\mu s + 5.12ns + 8\mu s) = 40.02\mu s$$

$$\text{After failure} : D_{e2e} = 6 * (2\mu s + 5.12ns + 8\mu s) = 60.03\mu s$$

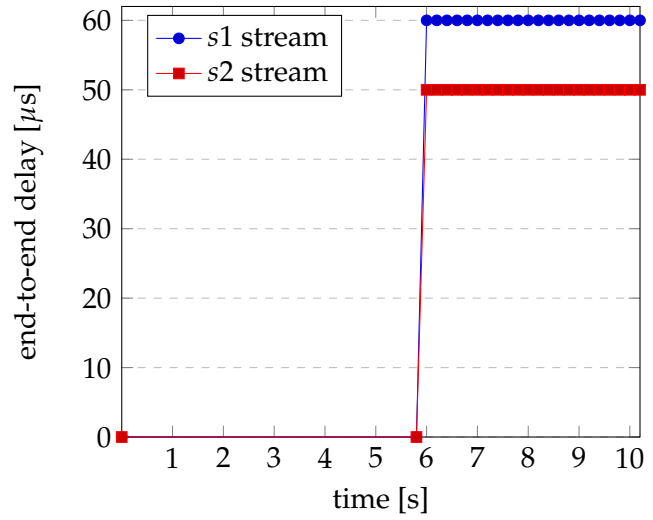
6.3.2.4 Testing FRER against Crash Failures

To evaluate the impact of crash failures, the fault injector sets the failure attribute of ETB switch 2. Consequently, ETB switch 2 stops forwarding frames to neighbour switches. After ETB switch two crashes (i.e. at $7.2th$ seconds), the HMI and monitoring application do not receive $s3$ anymore, because the only route of $s3$ passes through ETB switch 2. However, these applications continue receiving $s2$ from the redundant route which does not pass through the ETB switch 2. The crash failure results are presented in Figure 6.25.

It is good to mention that FRER cannot protect TSN networks from crash failures of all devices. For instance, when the fault injector sets the ECN switch 1 to a crash failure, no TT stream can flow between ECN1 and the other ECNs. ECN switch 1 is a single point of communication between the HMI and the other devices in the simulated network.



(A) Number of $s1$, $s2$ and $s3$ packets that are received by the HMI in case of l_1 failure



(B) End-to-end delay of $s1$ and $s2$ streams in the presence of $l1$ failure

FIGURE 6.24: Reliability of $s1$, $s2$ and $s3$ streams in case of $l1$ failure

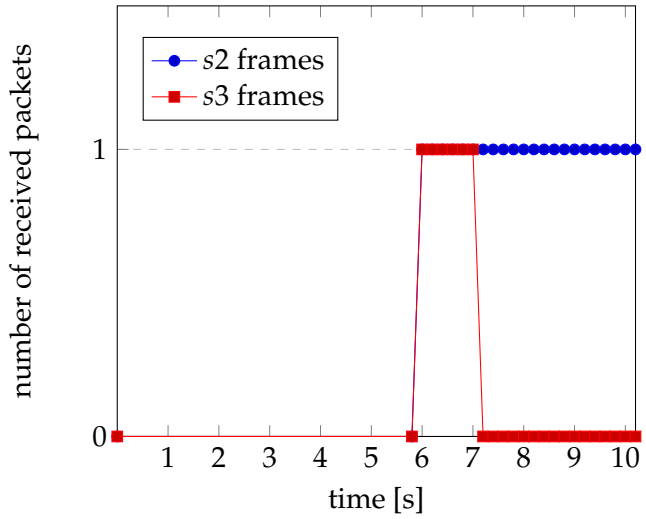


FIGURE 6.25: Number of $s2$ and $s3$ packets that are received by the HMI, and the monitoring application in case ETB switch two crashes

In the second part of the experiments, the behaviours of different TSN synchronization modules are studied thoroughly. To this end, several test scenarios are designed where each time-aware system is configured as follows:

- Announce message interval is configured to 3 seconds.
- Announce message time-out is set to 5 seconds.
- Synchronization message interval for the first five synchronization messages is set to 100 ms and after that, it is changed set to 1 second.
- The *Pdelay_Req* message is created right after reception of a synchronization message.
- A time-aware system sends the initial announce message at $5.2th$ seconds mainly because in the TSN simulator the gPTP messages are identified as BE traffic and this type of messages can only be sent after the RSTP convergence.
- The grandmaster clock dispatches the initial synchronization message at $5.3th$ seconds because BCMA needs to be executed first to elect a grandmaster gPTP device.
- In the experimental network, two priority attributes are defined for every time-aware system. Additionally, a drift rate of 500 ppm is configured for each system.

It has to be noted that the parameters described above are user-configurable. Therefore, they can be adjusted to meet the network requirements such as synchronization precision.

6.3.3 Time-Aware Network Simulation

As explained earlier immediately after the RSTP convergence, each time-aware system declares its pre-defined clock priorities to other systems through announce messages. The experimental setup comprises 23 time-aware systems. Therefore, every

time-aware system receives 22 announce messages from other systems within the simulated network. Each time-aware system upon reception of an announce message, executes the BMCA to specify its role as either a grandmaster or a slave clock.

All time-aware systems in the simulated network take part in the best master clock selection since they are all capable of taking the grandmaster role. In the experimental setup, HMI is marked as a grandmaster clock because it has the highest priority value among other devices. Once the execution of BMCA is completed, HMI begins to send synchronization and announce messages periodically. On the other hand, the rest of the time-aware systems which are slave clocks, schedule an announce message time-out event at which the operational state of the grandmaster clock is checked.

Once a slave clock receives a Sync message, it corrects its local time with the help of the grandmaster time embedded in a Sync message and the mean link delay. However, the peer delay mechanism is initiated upon reception of a Sync message. Hence, there is a time difference between the slave clock and the grandmaster clock during transmission of the first few Sync messages mainly because the mean link delay is not measured yet.

As described in section 3.5.5, the frequency offset between the grandmaster clock and slave clock (CSRO) is calculated using NRRs. A time-aware system calculates NRR each time it measures the mean path delay. After NRR computation, CSRO is calculated based on the principle depicted in Figure 3.21. It has to be noted that if NRR is calculated immediately after receiving a Sync message, the frequency offset between the grandmaster clock and slave clock would be negligible since the correction of the slave clock occurs.

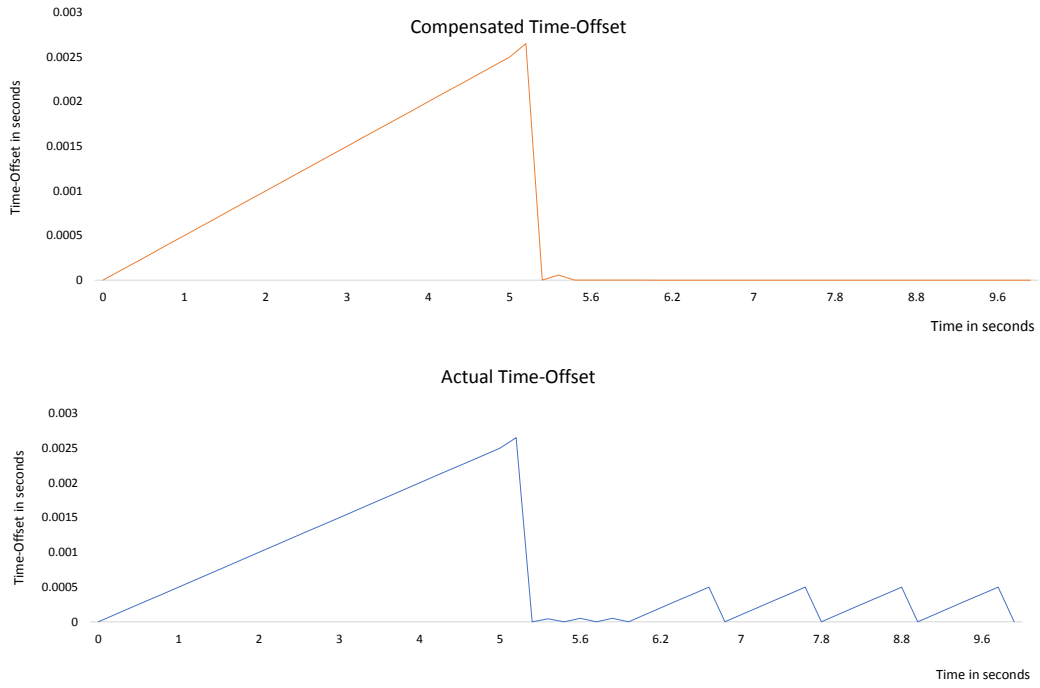


FIGURE 6.26: Time offset of sensor 3 with 500ppm drift rate

Figure 6.26 depicts the synchronization process of sensor 3 (i.e. slave gPTP device). As the graphs in Figure 6.26 show, the actual and the compensated time offset of sensor3 are the same until 5.3th seconds because sensor3 receives the first Sync message at that time instant. Therefore, during this time interval, the actual time

offset and compensated time offset remain the same because sensor3 has not calculated NRR, CSRO and link delay yet. After reception of the first five Sync messages, the compensated time offset converges to zero, which implies there is no time difference between the slave clock and the grandmaster clock. Thereby, it is important to configure the periodicity of Sync messages, particularly in the start-up interval so that the compensated time offset converges to zero faster. To this end, during the initialization phase the period of Sync messages is configured to 100 ms and after clock convergence, it changes to 1 s.

As mentioned earlier, in the emulated network, HMI is elected as a grandmaster clock, and other gPTP devices are marked as a slave clock. All slave gPTP devices behave similarly during the clock synchronization process. Therefore, the time offset of sensor3 is illustrated in Figure 6.26 as an example of a slave clock. It is noteworthy that the Sync messages traverse six intermediate time-aware bridges on the path from HMI to sensor3. Furthermore, the actual time-offset is reset to zero upon reception of a Sync message with the help of the measured link delay and CSRO. Then it increases linearly until the device receives the next Sync message.

6.3.4 Injecting Faults in a Time-aware Network

The above simulation run illustrates how the execution of BMCA, synchronization and link delay measurement result in clock convergence over the time-aware network. However, the mentioned scenario does not investigate the role of the Announce message time-out event in the synchronization process. To this end, first, the primary grandmaster clock fails, and then the behaviour of the compensated time offset is investigated during the failure and handover period.

In a time-aware network, a slave clock updates the Announce message time-out attribute upon reception of an Announce message from the grandmaster clock. On the other hand, if a slave gPTP device does not receive any Announce message during the Announce message time-out interval, it records the data of the current grandmaster clock for future rollback. It sends its clock information to other time-aware systems to initiate the BMCA mechanism. After BMCA execution, the new grandmaster clock replaces the current one until the primary grandmaster recovers. The primary grandmaster after recovery sends Announce messages to all time-aware systems. Therefore, all slave gPTP devices start synchronizing to the primary grandmaster clock. Moreover, the secondary grandmaster goes back to the slave state and stops sending Sync messages.

Like in the previous test scenario after initial BMCA execution, HMI is selected as a grandmaster while the other time-aware systems take the slave role and schedule an announce time-out event at $10th$ seconds. Further, the compensated time offset of all slave gPTP devices converges to zero after reception of the first five Sync messages and with the help of measured link delay and CSRO.

The fault injector fails the primary grandmaster for a time interval of approximately 7 seconds (i.e. from $6th$ seconds until $12.4th$ seconds). As illustrated in Figure 6.27, the compensated time-offset of sensor 3 remains zero throughout the grandmaster failure period, mainly because the clock frequency of sensor3 has been already locked with the clock frequency of the grandmaster. At $10th$ seconds, the announce time-out event is invoked in all slave clocks, including sensor3, which results in initiating BMCA. After BMCA execution, the monitoring application is marked as a new grandmaster and begins to send sync and Announce messages periodically towards other time-aware systems. As the graphs in Figure 6.27 show the compensated time-offset of sensor three does not change during grandmaster clock

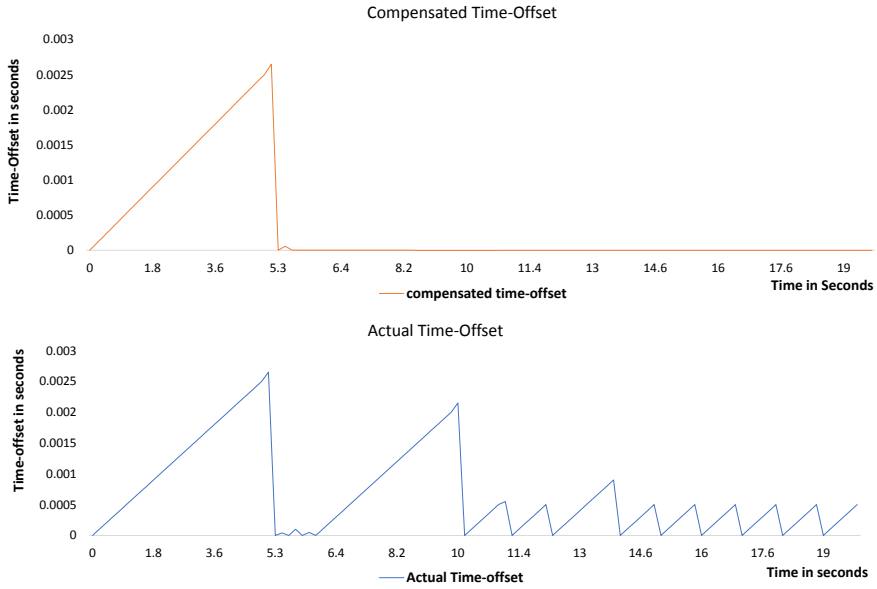


FIGURE 6.27: Time offset of sensor 3 with 500ppm drift rate when the primary grandmaster fails and recovers

handover since the local clock of a primary and newly elected grandmaster are identical (i.e. Riverbed simulation time) and therefore the frequency offset between the slave clock and the grandmaster clock (i.e. CSRO) remains constant. This is one the primary reason IEEE 802.1AS-Rev protocol is considered as a more robust clock synchronization mechanism compared to the existing synchronization methods.

HMI (i.e. the primary grandmaster) recovers at 12.4th seconds and starts sending Announce messages at 14th seconds because the Sync message interval is set to 1 second. All time-aware systems upon reception of a Sync message from HMI, rollback to the primary grandmaster clock properties. Moreover, the monitoring app (i.e. the current grandmaster) takes the slave role again and stops sending periodic Sync and Announce messages.

As explained earlier and shown in Equation 6.4, the local clock of a time-aware system is modelled with the linear drift rate. Therefore, the clock offset increases according to the time interval. The actual time offset of sensor3 as illustrated in Figure 6.27 increases linearly until 5.2th seconds and reaches its maximum value because sensor3 does not receive any Sync message during this period. After that, sensor3 receives a Sync message every 100 ms, which results in the correction of the clock.

Between 6th and 10th seconds, there is no grandmaster clock to send periodic Sync messages. Therefore, the graph in Figure 6.27 denotes that the actual time offset increases linearly during this period. Moreover, sensor3 does not receive any Sync messages between 12th seconds and 14th seconds due to grandmaster handover. As a result, Figure 6.27 depicts that the actual time offset of sensor3 during this period reaches a higher value compared to the other synchronization intervals.

Aside from the described use case, another test scenario is carried out where the link between HMI and the neighbour time-aware bridge (i.e. ECN switch1) fails.

The experimental results show the same behaviour as the above test scenario over the simulated time-aware network.

The computation of NRR for the local clock with linear drift rate is straightforward. However, this calculation could be challenging for a clock with an non-linear drift rate. To investigate this issue, the above simulation scenario is repeated with a non-linear clock model. Namely, the drift of the local clock of each time-aware system is modelled as follows [198]:

$$c(t) = \rho_0 t + \sigma t^2 \tag{6.12}$$

Where ρ_0 and σ correspond to a skew rate and clock drift rate respectively. For this simulation run, the skew rate is set to 0.01, and the drift rate is configured to 1000 ppm. We measure the compensated time offset for the local clock with linear and non-linear drift rates after completion of the synchronization process. In this duration, the maximum time offset of sensor3 clock from the grandmaster clock is 50 ns when the clock drift increases linearly while the maximum time offset is 200 ns for the clock with non-linear drift rate. For NRR computation, linearization is used. Therefore, there is a difference between the maximum compensated time offset of a clock with linear clock drift and non-linear drift rate. The non-linear clock drift could be linearized over a short period. Consequently, the synchronization accuracy for a non-linear clock could be improved if the synchronization interval is shortened. On the other hand, selecting the non-optimal periodicity for Sync messages leads to either an inefficient network usage or an inaccurate NRR. Therefore the synchronization interval needs to be chosen carefully by taking into account the clock drift and the required synchronization precision.

6.3.5 Time-aware Network with IEEE 802.1Qbv and Qci Integration

Each time-aware system within the simulated network models its local clock based on Equation 6.2. Therefore, before the convergence of clock synchronization, it is highly probable that the TT frames arrive outside the expected reception window and are dropped by the ingress time-based filtering module because there is a time difference among the clocks of different devices. In the TSN simulator, gPTP messages are classified as BE traffic. Therefore, ingress time-based filtering is not applied to gPTP messages. This design decision enables the establishment of a fully synchronized network, although, during the synchronization phase, all time-aware systems do not share the same notion of time. However, if gPTP messages are declared as TT frames, it is quite likely that at the beginning of the simulation, the time-based filtering module discards them due to clock drifts among different time-aware systems.

In the emulated network, an 802.1Qbv and Qci capable device filters incoming packets and schedules outgoing frames based on its own local time. Thereby, the transmission of TT frames should start after $6th$ seconds because time-aware systems start exchanging gPTP messages after $5.2th$ seconds. Moreover, they achieve the synchronization precision of sub microseconds after reception of five Sync message at approximately $6th$ seconds according to the simulation results from the above scenario. Figure 6.28 depicts the compensated time offset of CCU1 and the number of TT frames received by CCU1 from sensor 1 in the simulation run mentioned above. The experimental results showed that all frames sent from sensor1 reach CCU1 while the clock drifts of all time-aware systems, including sensor1 and CCU1, are corrected with the help of different gPTP mechanisms. In contrast, when sensor1 is configured to start sending TT frames towards CCU1 at the beginning of the simulation, CCU1 does not receive any frame from sensor1 according to the simulation results.

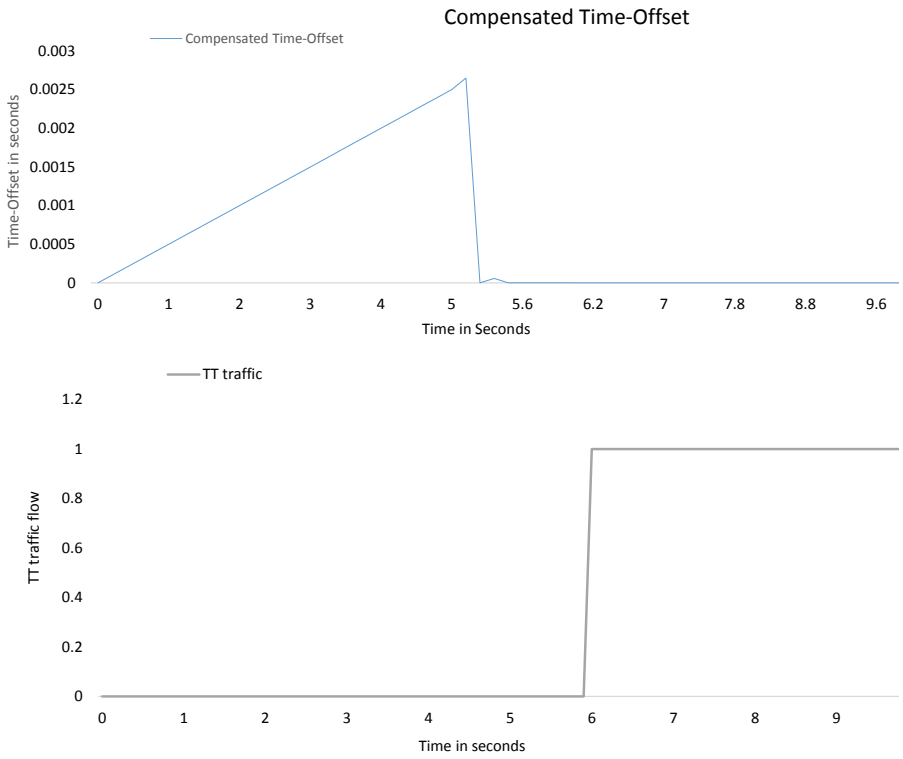


FIGURE 6.28: IEEE 802.1Qbv and Qci-capable device (i.e. CCU1) using gPTP local time for filtering and shaping traffic

The above simulation scenario is re-executed, but this time the local clock of all devices is directly obtained from the Riverbed simulation time instead of from Equation 6.2. Therefore, there are no clock drifts among TSN devices within the emulated network throughout the simulation run. The simulation results show the same trend as the previous use case, where there are clock drifts among gPTP devices.

To sum up, the empirical results show that the TSN centralized configuration model successfully collects configuration data from all network components, calculates the necessary configuration parameters and remotely deploys the computed information to the devices residing in the network. Therefore, dynamic configuration services in the TSN simulator provides an opportunity to simulate dynamic applications and system adaptation. Moreover, the experimental results illustrate that all devices in the simulated network achieve the synchronization accuracy in the order of sub microseconds using TSN time synchronization modules. Besides, the simulation results show that FRER modules offer reliable communication and protect the emulated network against transient and permanent errors. Furthermore, the experimental outputs depict that TSN time-based features provide the temporal isolation within the simulated system. Therefore, based on the empirical results we can conclude that TSN standard offers mixed-criticality traffic, reliable network services and the simplified configuration management, which are essential for the communication layer of the virtualized integration system.

Chapter 7

Conclusion

This chapter first summarizes the contributions of this thesis and then highlights potential future work.

7.1 Summary

Due to massive success and widespread deployment of Ethernet technologies, the Time-Sensitive Networking (TSN) task group introduces a series of IEEE 802.1 sub-protocols to offer a fault-tolerant and deterministic communication infrastructure for mission-critical systems over Ethernet-based networks. In TSN networks based on IEEE 802.1AS-Rev, all devices share a global clock which is realized using a fault-tolerant synchronization mechanism. Furthermore, TSN-aware components distinguish different message types in a stream (e.g. time-triggered and best effort) and store messages in separate egress queues. The TSN scheduling approach called Time-Aware Shaper (TAS) enforces the Gate Control List (GCL) of each port using a global notion of time. The GCL presents a transmission schedule table of all streams which are destined to a specific port.

This thesis introduces a GA-based procedure and a Heuristic List Scheduling strategy (HLS) for generating TT transmission schedules. In contrast to state-of-art scheduling solutions with fixed routing, the GA and HLS combines the routing and scheduling constraints and computes a system-wide schedule in a single-step. To make job binding and resource allocation feasible, a system model is specified in the form of an application graph and an architecture graph. Furthermore, in this thesis, the joint scheduling and routing constraints are defined and employed in the scheduling and optimization process of GA and HLS. Besides, these novel scheduling strategies support the distributed processing of real-time applications.

To have a solid base for comparison, a list scheduler is developed as an example of existing scheduling procedures which solve the routing and scheduling problems separately. The experimental results illustrate the impact of network load and system structure on GA, HLS and two-phased list scheduler performance indicators (i.e. scheduling capability and efficiency). GA and HLS extend the search space of scheduling possibilities using the joint constraints. Therefore, the scheduling capability of GA and HLS is enhanced significantly over a two-phased list scheduler, particularly under high network load. In the experiments, it is observed that GA and HLS improve the transmission makespan on average by 31 % and 39 % respectively compared to a two-phased list scheduler. The optimal makespan resulting from GA and HLS implies more compressed TT transmission schedules and a lower number of guard bands.

Moreover, this thesis presents a Fault-Tolerant GA (FTGA) and a Fault-Tolerant Heuristic List Scheduler (FTHLS) as extensions of TSN schedulers described earlier

with the primary goal of fulfilling the timing requirements of mission-critical applications while maximizing the system reliability. To achieve that, these strategies integrate TSN redundancy management mechanism including message duplication, message replica elimination and redundant paths selection into the scheduling process. Besides, the fault-tolerant TSN schedulers use a reliability analysis technique to compute the system reliability based on the reliability of each network component participating in message transmission. Therefore, these schedulers are different from the former fault-tolerant scheduling solutions that address either link failures or crash failures of devices. Apart from the novel reliability analysis technique, this work models the conditional control transfer between different safety-critical jobs using a conditional application graph. The conditional application graph provides an opportunity to formulate the precedence constraints of jobs more realistically and as a result, achieves more accurate schedulability analysis. The simulation results denote that FTGA and FTHLS improve the average of the system reliability compared to the basic GA and HLS, which does not support TSN redundancy management, at the expense of the makespan growth. This thesis additionally studies the impact of different component failure rates on system reliability through several synthetic system models. The experimental results illustrate that the reliability of the system is more sensitive to the failure rate of physical links compared to other components in the system. Furthermore, this work investigates the impact of redundant jobs on system reliability. The results demonstrate that the average of the system reliability is increased significantly as the number of substitutable TT messages grows.

The virtualized integrated system hosts several critical and non-critical modules that communicate with each other through a common networking infrastructure. Therefore, the communication layer of the integrated system shall provide reliable network services, mixed-criticality traffic, the simplified integration and configuration management. To achieve that, the TSN standard, which is the most recent real-time Ethernet extension and offers different features such as real-time capability, fault-tolerance and clock synchronization, is chosen as a networking solution for the virtualized integrated system. Simulation tools are seen as a cost and time-efficient approach to evaluate and verify network protocols, particularly during the development phase and before the actual implementation. This thesis presents a simulation framework for TSN, which is developed as an Ethernet-based network for mixed-critically traffic. The TSN simulation framework comprises the simulation models with both time-based and non-time-based services of TSN. Therefore, this work provides a more comprehensive simulation platform compared to the existing TSN simulators for modelling, performance and reliability evaluation of TSN networks. Moreover, this thesis uses an example layout of an Ethernet-based train network to evaluate the TSN simulation framework on configuration, determinism, fault-tolerance and clock synchronization because this simulated network provides a more realistic setup for studying different aspects of TSN features.

In more details, the TSN simulation framework contains the fully centralized configuration model, as proposed in IEEE 802.1Qcc. This configuration model provides an efficient way to centralize the network resource allocation, the computations of transmission schedules and other necessary configuration information. Besides, the central entity sends the NETCONF messages carrying configuration data to TSN devices. The NETCONF standard, however, offers several features which are essential for managing the complex real-time system, but it does not satisfy the stringent timing and reliability requirements of many safety-critical systems. The reason is that NETCONF transmits messages based on the best-effort paradigm.

Aside from the central configuration model, the TSN simulation framework comprises the simulation models of TSN switches and end systems. These simulation models implement the egress time-aware shaping and ingress time-based filtering modules which are key enablers for temporal isolation over TSN networks. Furthermore, the simulation models develop different FRER functionalities to support reliable communication. The fault-tolerance capability of TSN devices is evaluated using a highly redundant train network. The simulation results show that the IEEE 802.1CB standard protects time-sensitive systems against transient errors (e.g. stuck transmitter, resequencing) and offers bounded end-to-end delay and zero packet loss in case of permanent errors (e.g. link failure, node crash). The simulation models additionally incorporate TSN clock synchronization to establish and maintain the synchronized global clock within the network. The empirical results illustrate that the TSN synchronization process offers the synchronization accuracy of one microsecond even in the presence of other traffic in the emulated network. Nevertheless, the synchronization precision highly depends on the clock drift rate, the synchronization interval, the accuracy of NRR calculation, the announce time-out and the accuracy of the measured timestamps.

7.2 Future Work

The studies carried out in this thesis can be extended in many aspects. Therefore, potential future work is listed as follows:

1. The scheduling strategies proposed in this thesis compute a valid GCL in a way that the AVB streams and BE traffic which do not have strict timing requirements, do not interfere with the transmission of TT frames. Consequently, these schedulers only compute the static transmission schedule tables of TT messages, and non-TT frames (including AVB and BE traffic) are sent when no TT message is scheduled. Therefore, the described schedulers ignore the impact of TT transmission schedules on the end-to-end delay of AVB streams which may result in loss of AVB messages. To address this issue, the AVB stream requirements need to be considered during the routing and scheduling process of TT traffic. Moreover, the main goal of the scheduling strategies shall satisfy the deadlines of real-time applications while optimizing TT transmission makespan and delivery delay of AVB streams.

2. The fault-tolerant schedulers proposed in this thesis only focus on permanent failures and calculate the system reliability based on the hardware reliability model. Thereby, these scheduling strategies can be extended to address intermittent and transient failures according to a more realistic reliability model of the network components.

3. The simulated TSN network only consists of a single gPTP domain. However, the IEEE 802.1As-Rev standard supports multiple gPTP domains. Hence, the simulation framework can be extended so that it could emulate a network with multiple gPTP domains.

4. The IEEE 802.1As-Rev standard permits multiple time scales within a single gPTP domain. However, the TSN simulation framework does not contain this feature. Therefore, the TSN simulator can be extended to support multiple time scales.

5. In gPTP systems, time-aware systems can interconnect via four different types of links. Nonetheless, the simulated time-aware network only uses full-duplex Ethernet links for interconnecting devices. The experimental setup can be modified to include other link technologies. Besides, the simulation models need to implement the delay measurement mechanism associated with each transmission link.

6. IEEE 802.1Qcc proposes three different configuration models. Nevertheless, the TSN simulation framework only deploys the centralized configuration model for network management and configuration purposes. Therefore, future work can implement other TSN configuration models and further evaluate the impact of these configuration model on the emulated network performance.

7. Last but not least, the correctness and applicability of the simulation results should be verified by conducting the same experiments on the setup with actual TSN devices and Ethernet links.

Bibliography

[1] R. Obermaisser, C. El Salloum, B. Huber, and H. Kopetz, "From a Federated to an Integrated Automotive Architecture", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 7, pp. 956–965, 2009.

[2] H. Kopetz, "An Integrated Architecture for Dependable Embedded Systems", in *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, IEEE, 2004, pp. 160–161.

[3] A. Grasset, "Design of Critical Embedded Systems: from Early Specifications to Prototypes", in *2015 International Symposium on Rapid System Prototyping (RSP)*, IEEE, 2015, pp. 38–38.

[4] A. Specification, "651: Design Guidance for Integrated Modular Avionics", *Aeronautical Radio, Inc, Annapolis, MD*, 1991.

[5] G. AUTOSAR, "AUTOSAR Technical Overview", *Technical Overview Version*, vol. 2, no. 1, 2006.

[6] M. Jakovljevic, A. Geven, N. Simanic-John, and D. Saatci, "Next-Gen Train Control/Management (TCMS) Architectures:"Drive-By-Data" System Integration Approach", in *2018 9th European Congress Embedded Real-Time Software and Systems*, 2018.

[7] M. J. e. all, "State-Of-The-Art Document on Drive-by-Data", European Union, Tech. Rep., 2016, Safe4Rail project. [Online]. Available: https://safe4rail-1.safe4rail.eu/downloads/deliverables/Safe4RAIL-D1.1-State-of-the-Art-Drive-by-Data-PU-M3.pdf (visited on 03/10/2019).

[8] R. Fuchsen, "IMA NextGen: a New Technology for the Scarlett Program", *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 10, pp. 10–16, 2010.

[9] "ASHLEY PROJECT", European Aerospace Industry, Tech. Rep., 2016. [Online]. Available: http://www.ashleyproject.eu/the-project-summary (visited on 03/10/2019).

[10] A. Draft, *of Project Paper 664: Aircraft Data Network, Part 7-Avionics Full Duplex Switched Ethemet (AFDX) Network*, 3.

[11] "Institute of Electrical and Electronics Engineers, Time-Sensitive Networking", in *Time-Sensitive Networking Task Group*, IEEE, 2017. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html.

[12] "Institute of Electrical and Electronics Engineers, Inc. 802.1Qbv - Enhancements for Scheduled Traffic", in *Time-Sensitive Networking Task Group*, IEEE, 2016. [Online]. Available: http://www.ieee802.org/1/pages/802.1bv.html.

[13] "Institute of Electrical and Electronics Engineers, Audio/Video Bridging", in *The Audio/Video Bridging Task Group*, IEEE, 2011. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html.

[14] M. J. e. all, "Initial Drive-by-Data Draft Concept Design", European Union, Tech. Rep., 2016, SAfe4Rail project. [Online]. Available: https://safe4rail-1.safe4rail.eu/downloads/deliverables/Safe4RAIL-D1.3-Initial-Drive-by-Data-Draft-Concept-Design-PU-M6.pdf (visited on 03/10/2019).

[15] J.-D. Decotignie, "Ethernet-based real-time and industrial communications", *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1102–1117, 2005.

[16] M. W. et al., "Time-aware applications, computers, and communication systems", Tech. rep. 2015, Technical Note (NIST TN)-1867.

[17] "Institute of Electrical and Electronics Engineers, Inc. 802.1AS-Rev - Timing and Synchronization for Time-Sensitive Applications", in *Time-Sensitive Networking Task Group.*, IEEE, 2017. [Online]. Available: http://www.ieee802.org/1/pages/802.1AS-rev.html.

[18] M. Pahlevan and R. Obermaisser, "Evaluation of Time-Triggered Traffic in Time Sensitive Networks using the OPNET Simulation Framework", in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, IEEE, 2018, pp. 283–287.

[19] M. Pahlevan and R. Obermaisser, "Redundancy Management for Safety Critical Applications with Time Sensitive Networking", in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, IEEE, 2018, pp. 1–7.

[20] M. Pahlevan, B. Balakrishna, and R. Obermaisser, "Simulation Framework for Clock Synchronization in Time Sensitive Networking", in *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, 2019, pp. 213–220.

[21] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks", *ACM Sigbed Review*, vol. 16, no. 1, pp. 15–20, 2019.

[22] M. Pahlevan and R. Obermaisser, "Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks", in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, vol. 1, 2018, pp. 337–344.

[23] M. Pahlevan, S. Amin, and R. Obermaisser, "Fault-Tolerant List Scheduler for Time-Triggered Communication in Time-Sensitive Networks", in *2019 4th International Conference on System Reliability and Safety*, IEEE, 2019.

[24] "Introduction to Riverbed Modeler Academic Edition", 2018. [Online]. Available: https:splash.riverbed.com/docs/DOC-4833.

[25] "Institute of Electrical and Electronics Engineers, Inc. 802.1Qci -Per-Stream Filtering and Policing", in *Time-Sensitive Networking Task Group*, IEEE, 2016. [Online]. Available: http://www.ieee802.org/1/pages/802.1ci.html.

[26] "Institute of Electrical and Electronics Engineers,P802.1Qcc – Stream Reservation Protocol (SRP) Enhancements and Performance Improvements, Draft 1.6", in *Time-Sensitive Networking Task Group*, IEEE, 2017. [Online]. Available: http://www.ieee802.org/1/files/private/cc-drafts/d1/802-1Qcc-d1-6.pdf.

[27] M. Pahlevan, J. Schmeck, and R. Obermaisser, "Evaluation of TSN Dynamic Configuration Model for Safety-Critical Applications", in *2018 17th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, IEEE, 2019.

[28] "Institute of Electrical and Electronics Engineers, Inc. 802.1CB - Frame Replication and Elimination for Reliability", in *Time Sensitive Networking Task Group*, IEEE, 2017. [Online]. Available: http://www.ieee802.org/1/files/private/cb-drafts/d2/802-1CB-D2-9.pdf.

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

[30] R. Obermaisser, *Time-triggered communication*. CRC Press, 2011.

[31] F. Consortium *et al.*, *FlexRay communications system protocol specification version 2.1*, 2005.

[32] H. Kopetz and G. Bauer, "The time-triggered architecture", *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[33] M. D. Mesarovic and Y. Takahara, "Abstract systems theory", 1989.

[34] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg, and H. Sivencrona, "The real byzantine generals", in *The 23rd Digital Avionics Systems Conference (IEEE Cat. No. 04CH37576)*, IEEE, vol. 2, 2004, pp. 6–D.

[35] "IEEE Standard for Local and Metropolitan Area Networks:Overview and Architecture", in *LAN/MAN Standards Committee*, IEEE, 2014.

[36] "IEC 61158 Type 3 - PROFIBUS Standard", in *International Electrotechnical Commission*, IEC, 2014.

[37] "ISO 11898-1, Controller area network (CAN)", in *ISO/TC 22/SC 31 Data communication*, ISO, 2018.

[38] N. Warden, "Overview and effect of deterministic ethernet on test strategies", in *2017 IEEE AUTOTESTCON*, IEEE, 2017, pp. 1–3.

[39] "Institute of Electrical and Electronics Engineers, IEEE Std 802.3-1985", in *LAN/MAN Standards Committee*, IEEE, 1985.

[40] "IEEE 802.3u IEEE Standards for Local and Metropolitan Area Networks: Supplement-Media Access Control Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100Mb/s Operation, Type 100BASE-T", in *LAN/MAN Standards Committee*, IEEE, 1995.

[41] "IEEE 802.3z-1998 - Media Access Control Parameters, Physical Layers, Repeater and Management Parameters for 1,000 Mb/s Operation, Supplement to Information Technology - Local and Metropolitan Area Networks", in *LAN/MAN Standards Committee*, IEEE, 1998.

[42] "IEEE 802.3ab-1999 - Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Physical Layer Parameters and Specifications for 1000 Mb/s Operation", in *LAN/MAN Standards Committee*, IEEE, 1999.

[43] "Institute of Electrical and Electronics Engineers, IEEE Std 802.3™-2018", in *LAN/MAN Standards Committee*, IEEE, 2018.

[44] R. Cole, "An introduction to packet switched computer networks", *Science Progress (1933-)*, pp. 127–142, 1982.

[45] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique", *Computer Networks (1976)*, vol. 3, no. 4, pp. 267–286, 1979.

[46] M. Ilyas and S. Bhatia, "Cut-through switching for integrated services packet networks", in *[1988] Proceedings. Computer Networking Symposium*, IEEE, 1988, pp. 405–410.

[47] "IEEE Std 802.1D™, IEEE Standards for Local and metropolitan area networks - Media Access Control (MAC) Bridges", in *LAN/MAN Standards Committee*, IEEE, 2003.

[48] "IEEE Std 802.1Q™, IEEE Standards for Local and metropolitan area networks - Virtual Bridged Local Area Networks", in *LAN/MAN Standards Committee*, IEEE, 2003.

[49] "IEEE Std 802.1D, IEEE Standard for Local and metropolitan area networks Media Access Control (MAC) Bridges", IEEE, 2004.

[50] "IEC 62439-2 Ed.01, Industrial communication networks - High availability automation networks - Part 2: Media Redundancy Protocol (MRP)", IEC, 2010.

[51] "IEC 62439-3 Ed.03, Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR)", IEC, 2016.

[52] "IEC 62439-4 Ed.02, Industrial communication networks - High availability automation networks - Part 4: Cross-network Redundancy Protocol", IEC, 2010.

[53] "IEC 62439-5 Ed.02, Industrial communication networks - High availability automation networks - Part 5: Beacon Redundancy Protocol", IEC, 2016.

[54] "IEC 62439-6 Ed.01, Industrial communication networks - High availability automation networks - Part 6: Distributed Redundancy Protocol", IEC, 2010.

[55] "IEC 62439-7 Ed.01, Industrial communication networks - High availability automation networks - Part 7: Ring-based Redundancy Protocol", IEC, 2012.

[56] A. Giorgetti, F. Cugini, F. Paolucci, L. Valcarenghi, A. Pistone, and P. Castoldi, "Performance analysis of media redundancy protocol (MRP)", *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 218–227, 2013.

[57] H. Kirrmann, "Highly Available Automation Networks Standard Redundancy Methods -Rationales behind the IEC 62439 standard suite", ABB Switzerland Ltd, Tech. Rep., 2012. [Online]. Available: http://caxapa.ru/thumbs/767218/IEC_62439_Summary.pdf (visited on 04/10/2019).

[58] J. Araujo, J Lazaro, A Astarloa, A Zuloaga, and A Garcia, "PRP and HSR version 1 (IEC 62439-2), improvements and a prototype implementation", in *IECON 2013-39th Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2013, pp. 4410–4415.

[59] C. Hoga, "Seamless communication redundancy of IEC 62439", in *2011 International Conference on Advanced Power System Automation and Protection*, IEEE, vol. 1, 2011, pp. 489–494.

[60] "Institute of Electrical and Electronics Engineers, IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE, 2008. [Online]. Available: https://standards.ieee.org/findstds/interps/1588-2008.html.

[61] L. Han-Rong, W. Wei-Jiang, Z. Feng, and L. Li, "Recovery Time Analysis of a Distributed Redundancy Protocol", in *Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering*, Springer, 2013, pp. 9–15.

[62] G. Yoon, S.-G. Lee, D.-H. Kwon, S.-C. Kwon, and Y.-O. Park, "RAPIEnet based redundancy control system", in *2011 11th International Conference on Control, Automation and Systems*, IEEE, 2011, pp. 140–145.

[63] "Internet Engineering Task Force, Network Time Protocol", IETF, 2010. [Online]. Available: https://tools.ietf.org/html/rfc5905.

[64] "Institute of Electrical and Electronics Engineers, Inc. 802.1AS - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks", in *Time-Sensitive Networking Task Group.*, IEEE, 2010. [Online]. Available: http://www.ieee802.org/1/files/private/as-drafts/d7/802-1AS-d7-6.pdf.

[65] "IEEE 1588 Precision Time Synchronization Solution for Electric Utilities", Siemens AG, Tech. Rep., 2011. [Online]. Available: http://www.fujitsu.com/downloads/TEL/fnc/pdfservices/ethernet-prerequisite.pdf.

[66] B. Balakrishna, *Developing a simulation platform for Time-Sensitive Networking*, Germany, 2019.

[67] M. D. Pardue, "Fine-tuning the osi model: Layer functions and services", in *MILCOM 1987-IEEE Military Communications Conference-Crisis Communications: The Promise and Reality*, IEEE, vol. 1, 1987, pp. 0199–0203.

[68] "Implementing IEEE 1588v2 for use in the mobile backhaul", Calnex Solutions Ltd, Tech. Rep., 2009.

[69] M. Beck, *Ethernet in the First Mile: the IEEE 802.3 ah EFM standard*. McGraw Hill Professional, 2005.

[70] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai, "IEEE 802.11 wireless local area networks", *IEEE Communications magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[71] K. Matheus and T. Königseder, *Automotive ethernet*. Cambridge University Press, 2017.

[72] J. Postel, "Internet protocol", 1981.

[73] R. Hinden, "Internet protocol, version 6 (IPv6) specification", 2017.

[74] PROFIBUS & PROFINET International, "Profinet system description technology and application", PROFIBUS & PROFINET International, Tech. Rep., 2014.

[75] A. E. E. Committee *et al.*, "Aircraft Data Network Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network, ARINC Specification 664", *Aeronautical Radio*, 2005.

[76] S. AS6802, "Time-Triggered Ethernet", *SAE International*, 2011.

[77] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer, "The time-triggered ethernet (TTE) design", in *Eighth IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC'05)*, IEEE, 2005, pp. 22–33.

[78] W. Steiner and G. Bauer, "TTethernet: Time-triggered services for ethernet networks", in *Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th*, 2009, p. 1.

[79] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks", *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 86–94, 2016.

[80] "Audio Video Bridging (AVB)", Arista Networks, Tech. Rep., 2009.

[81] "IEEE P802.1Qat/D6.1 - Stream Reservation Protocol (SRP)", in *Audio/Video Bridging Task Group*, IEEE, 2010. [Online]. Available: https://standards.ieee.org/findstds/standard/802.1Qat-2010.html.

[82] "IEEE P802.1Qav/D7.0 - Forwarding and Queuing Enhancements for Time-Sensitive Streams", in *Audio/Video Bridging Task Group*, IEEE, 2009. [Online]. Available: https://standards.ieee.org/findstds/standard/802.1Qav-2009.html.

[83] "IEEE Standard for Local and metropolitan area networks— Audio Video Bridging (AVB) Systems", in *Audio/Video Bridging Task Group*, IEEE, 2011. [Online]. Available: http://standards.ieee.org/getieee802/download/802.1BA-2009.html.

[84] Cisco, "Cisco Audio Video Bridging Design and Deployment for Enterprise Networks", Cisco, Tech. Rep., 2018.

[85] "IEEE P802.1ak/D8.0 - Multiple Registration Protocol (SRP)", in *Interworking Task Group of IEEE 802.1*, IEEE, 2006. [Online]. Available: http://www.ieee802.org/1/files/private/ak-drafts/d8/802-1ak-d8-0.pdf.

[86] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic", in *2013 IEEE Vehicular Networking Conference*, IEEE, 2013, pp. 47–54.

[87] E. Heidinger, F. Geyer, S. Schneele, and M. Paulitsch, "A performance study of Audio Video Bridging in aeronautic Ethernet networks", in *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, IEEE, 2012, pp. 67–75.

[88] F. Frances, C. Fraboul, and J. Grieu, "Using network calculus to optimize the AFDX network", 2006.

[89] R. Queck, "Analysis of Ethernet AVB for automotive networks using Network Calculus", in *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*, IEEE, 2012, pp. 61–67.

[90] L. Zhao, P. Pop, Z. Zheng, and Q. Li, "Timing analysis of AVB traffic in TSN networks using network calculus", in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2018, pp. 25–36.

[91] L. Zhao, F. He, and J. Lu, "Comparison of AFDX and audio video bridging forwarding methods using network calculus approach", in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, IEEE, 2017, pp. 1–7.

[92] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach", *IEEE Transactions on Industrial informatics*, vol. 6, no. 4, pp. 521–533, 2010.

[93] L. Zhao, H. Xiong, Q. Li, and F. He, "Using memo recursive computation in the Trajectory approach for the worst-case delay analysis of AFDX networks", in *2011 International Conference on Electrical and Control Engineering*, IEEE, 2011, pp. 5633–5638.

[94] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Worst-case backlog evaluation of avionics switched ethernet networks with the trajectory approach", in *2012 24th Euromicro Conference on Real-Time Systems*, IEEE, 2012, pp. 78–87.

[95] X. Li, O. Cros, and L. George, "The Trajectory approach for AFDX FIFO networks revisited and corrected", in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, 2014, pp. 1–10.

[96] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks", in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ACM, 2016, pp. 183–192.

[97] e. a. Marc Weiss, "Time-aware applications, computers, and communication systems (TAACCS)", Technical Note (NIST TN)-1867, Tech. Rep., 2015. [Online]. Available: http://www.fujitsu.com/downloads/TEL/fnc/pdfservices/ethernet-prerequisite.pdf.

[98] "Institute of Electrical and Electronics Engineers, Inc. 802.1Qca - Path Control and Reservation", in *Time-Sensitive Networking Task Group*, IEEE, 2015.

[99] "Internet Engineering Task Force, the YANG 1.1 Data Modeling Language", IETF, 2016. [Online]. Available: https://tools.ietf.org/html/rfc7950#section-45.

[100] "Internet Engineering Task Force, Network Configuration Protocol", IETF, 2011. [Online]. Available: https://tools.ietf.org/html/rfc6241#section-4.5.

[101] "Network Working Group, Uniform Resource Identifier (URI): Generic Syntax", NWG, 2005. [Online]. Available: https://tools.ietf.org/html/std66.

[102] J. Schmeck, *Dynamic configuration for safety-critical applications in TSN networks*, Germany, 2019.

[103] "Extensible Markup Language (XML) 1.0", W3C, 2000. [Online]. Available: https://www.w3.org/TR/2000/REC-xml-20001006-review.html.

[104] B. J. Nelson, "Remote Procedure Call", XEROX, Tech. Rep., 1981.

[105] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks", in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, IEEE, 2017, pp. 1–6.

[106] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks", in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, ACM, 2017, pp. 8–17.

[107] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks", in *2010 31st IEEE Real-Time Systems Symposium*, IEEE, 2010, pp. 375–384.

[108] W. Steiner, "Synthesis of static communication schedules for mixed-criticality systems", in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, IEEE, 2011, pp. 11–18.

[109] D. Tamas-Selicean, P. Pop, and W. Steiner, "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems", in *Proceedings of the 8th IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ACM, 2012, pp. 473–482.

[110] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems", *Real-Time Systems*, vol. 51, no. 1, pp. 1–35, 2015.

[111] L. Bingqian and W. Yong, "Hybrid-GA based static schedule generation for time-triggered ethernet", in *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, IEEE, 2016, pp. 423–427.

[112] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task-and network-level schedule co-synthesis of Ethernet-based time-triggered systems", in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2014, pp. 119–124.

[113] S. S. Craciunas and R. S. Oliver, "SMT-based task-and network-level static schedule generation for time-triggered networked systems", in *Proceedings of the 22nd international conference on real-time networks and systems*, ACM, 2014, p. 45.

[114] S. S. Craciunas and R. S. Oliver, "Combined task-and network-level scheduling for distributed time-triggered systems", *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016.

[115] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, "Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing", in *2017 IEEE Fog World Congress (FWC)*, IEEE, 2017, pp. 1–6.

[116] A. M. Kentis, M. S. Berger, and J. Soler, "Effects of port congestion in the gate control list scheduling of time sensitive networks", in *2017 8th International Conference on the Network of the Future (NOF)*, IEEE, 2017, pp. 138–140.

[117] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)", in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ACM, 2016, pp. 203–212.

[118] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop, "AVB-aware routing and scheduling of time-triggered traffic for TSN", *Ieee Access*, vol. 6, pp. 75 229–75 243, 2018.

[119] J. Y. Yen, "Finding the k shortest loopless paths in a network", *management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[120] M. G. Resende and C. C. Ribeiro, "GRASP: Greedy randomized adaptive search procedures", in *Search methodologies*, Springer, 2014, pp. 287–312.

[121] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications", in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ACM, 2016, pp. 193–202.

[122] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for tsn with ilp", in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, IEEE, 2018, pp. 136–146.

[123] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich, "Combined system synthesis and communication architecture exploration for MPSoCs", in *Proceedings of the Conference on Design, Automation and Test in Europe*, European Design and Automation Association, 2009, pp. 472–477.

[124] "GAlib Documentation", in *http://lancet.mit.edu/galib-2.4/*, 2017.

[125] "SNAP Library 4.0, User Reference Documentation", 2017. [Online]. Available: `https://snap.stanford.edu/snap/doc/snapuser-ref/index.html`.

[126] I. Koren and C. M. Krishna, *Fault-tolerant systems*. Elsevier, 2010.

[127] G. Avni, S. Guha, and G. Rodriguez-Navas, "Synthesizing time triggered schedules for switched networks with faulty links", in *Embedded Software (EMSOFT), 2016 International Conference on*, IEEE, 2016, pp. 1–10.

[128] C. Dima, A. Girault, C. Lavarenne, and Y. Sorel, "Off-line real-time fault-tolerant scheduling", in *Proceedings Ninth Euromicro Workshop on Parallel and Distributed Processing*, IEEE, 2001, pp. 410–417.

[129] D. Mosse, R. Melhem, and S. Ghosh, "Analysis of a fault-tolerant multiprocessor scheduling algorithm", in *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*, IEEE, 1994, pp. 16–25.

[130] H. Lee, J. Kim, and S. J. Hong, "Evaluation of two load-balancing primary-backup process allocation schemes", *IEICE TRANSACTIONS on Information and Systems*, vol. 82, no. 12, pp. 1535–1544, 1999.

[131] R. Al-Omari, G. Manimaran, and A. Somani, "A Fault–Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real–Time Systems", in *Fault-tolerant Computing Symp.(FTCS), FAST ABSTRACTS*, 1999, pp. 63–64.

[132] X. Qin, H. Jiang, and D. R. Swanson, "A fault-tolerant real-time scheduling algorithm for precedence-constrained tasks in distributed heterogeneous systems", *Technical Report TR-UNL-CSE 2001–1003*, 2001.

[133] A. Benoit, M. Hakem, and Y. Robert, "Contention awareness and fault-tolerant scheduling for precedence constrained tasks in heterogeneous systems", *Parallel Computing*, vol. 35, no. 2, pp. 83–108, 2009.

[134] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems", in *Design, Automation and Test in Europe, 2005. Proceedings*, IEEE, 2005, pp. 864–869.

[135] X. Cui, J. Zhang, K. Wu, and E. Sha, "Efficient feasibility analysis of DAG scheduling with real-time constraints in the presence of faults", in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, IEEE, 2014, pp. 131–136.

[136] L. Su, H. Wan, Y. Qin, X. Zhao, Y. Gao, X. Song, C. Lu, and M. Gu, "Synthesizing Fault-Tolerant Schedule for Time-Triggered Network Without Hot Backup", *IEEE Transactions on Industrial Electronics*, vol. 66, no. 2, pp. 1345–1355, 2018.

[137] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-resilient topology planning and traffic configuration for ieee 802.1 qbv tsn networks", in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, IEEE, 2018, pp. 151–156.

[138] S. M. Shatz, J.-P. Wang, and M. Goto, "Task allocation for maximizing reliability of distributed computer systems", *IEEE Transactions on Computers*, vol. 41, no. 9, pp. 1156–1168, 1992.

[139] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 308–323, 2002.

[140] F. Smirnov, F. Reimann, J. Teich, Z. Han, and M. Glaß, "Automatic optimization of redundant message routings in automotive networks", in *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems*, ACM, 2018, pp. 90–99.

[141] "IEC 61375-2-5:2014. Electronic railway equipment - train communication network (TCN) - part 2-5: Ethernet train backbone.", in *International Electrotechnical Commission*, IEC, 2014.

[142] A. Benoit, F. Dufossé, A. Girault, and Y. Robert, "Reliability and performance optimization of pipelined real-time systems", *Journal of Parallel and Distributed Computing*, vol. 73, no. 6, pp. 851–865, 2013.

[143] D. E. Goldberg, *Genetic algorithms*. Pearson Education India, 2006.

[144] O. Sinnen, *Task scheduling for parallel systems*. John Wiley & Sons, 2007, vol. 60.

[145] P. Heise, F. Geyer, and R. 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*, IEEE, 2016, pp. 1–5.

[146] H. Kirrmann, M. Hansson, and P. Muri, "IEC 62439 PRP: Bumpless recovery for highly available, hard real-time industrial networks", in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, IEEE, 2007, pp. 1396–1399.

[147] H. Kirrmann, K. Weber, O. Kleineberg, and H. Weibel, "Seamless and low-cost redundancy for substation automation systems (high availability seamless redundancy, HSR)", in *2011 IEEE Power and Energy Society General Meeting*, IEEE, 2011, pp. 1–7.

[148] S. A. Nsaif and J. M. Rhee, "Seamless ethernet approach", in *IEEE International Conference on Consumer Electronics (ICCE), 2016*, IEEE, 2016, pp. 385–388.

[149] I. e. all, "TCMS Framework Concept", European Union, Tech. Rep., 2017, Safe4Rail project. [Online]. Available: https://safe4rail-1.safe4rail.eu/downloads/deliverables/Safe4RAIL-D2.3-TCMS-framework-concept-PU-M18-with-Annexes.pdf (visited on 03/10/2019).

[150] G. Alderisi, G. Iannizzotto, and L. L. Bello, "Towards IEEE 802.1 Ethernet AVB for advanced driver assistance systems: A preliminary assessment", in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, IEEE, 2012, pp. 1–4.

[151] H. Zinner, J. Noebauer, T. Gallner, J. Seitz, and T. Waas, "Application and realization of gateways between conventional automotive and IP/ethernet-based networks", in *Proceedings of the 48th Design Automation Conference*, ACM, 2011, pp. 1–6.

[152] T. Steinbach, H. D. Kenfack, F. Korf, and T. C. Schmidt, "An extension of the OMNeT++ INET framework for simulating real-time ethernet with high accuracy", in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, ICST (Institute for Computer Sciences, Social-Informatics, 2011, pp. 375–382.

[153] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's in-car interconnect? A competitive evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)", in *2012 IEEE Vehicular Technology Conference (VTC Fall)*, IEEE, 2012, pp. 1–5.

[154] G. Alderisi, A. Caltabiano, G. Vasta, G. Iannizzotto, T. Steinbach, and L. L. Bello, "Simulative assessments of IEEE 802.1 Ethernet AVB and time-triggered Ethernet for advanced driver assistance systems and in-car infotainment", in *2012 IEEE Vehicular Networking Conference (VNC)*, IEEE, 2012, pp. 187–194.

[155] K. Kawahara, Y. Matsubara, and H. Takada, "A Simulation Environment and preliminary evaluation for Automotive CAN-Ethernet AVB Networks", *arXiv preprint arXiv:1409.0998*, 2014.

[156] S. Tuohy, M. Glavin, C. Hughes, E. Jones, and L. Kilmartin, "An ns-3 based simulation testbed for in-vehicle communication networks", in *27th Annual UK Performance Engineering Workshop, Bradford, UK*, 2011.

[157] J. Imtiaz, J. Jasperneite, and L. Han, "A performance study of Ethernet Audio Video Bridging (AVB) for Industrial real-time communication", in *2009 IEEE Conference on Emerging Technologies & Factory Automation*, IEEE, 2009, pp. 1–8.

[158] J. Imtiaz, J. Jasperneite, and K. Weber, "Approaches to reduce the latency for high priority traffic in IEEE 802.1 AVB networks", in *2012 9th IEEE International Workshop on Factory Communication Systems*, IEEE, 2012, pp. 161–164.

[159] H.-T. Lim, D. Herrscher, M. J. Waltl, and F. Chaari, "Performance analysis of the ieee 802.1 ethernet audio/video bridging standard", in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ICST (Institute for Computer Sciences, Social-Informatics, 2012, pp. 27–36.

[160] H.-T. Lim, D. Herrscher, L. Völker, and M. J. Waltl, "IEEE 802.1 AS time synchronization in a switched Ethernet based in-car network", in *Vehicular Networking Conference (VNC), 2011 IEEE*, IEEE, 2011, pp. 147–154.

[161] G. Alderisi, G. Patti, and L. L. Bello, "Introducing support for scheduled traffic over IEEE audio video bridging networks", in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, IEEE, 2013, pp. 1–9.

[162] C. Park, J. Lee, T. Tan, and S. Park, "Simulation of scheduled traffic for the IEEE 802.1 time sensitive networking", in *Information Science and Applications (ICISA) 2016*, Springer, 2016, pp. 75–83.

[163] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang, "A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++", in *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, IEEE, 2018, pp. 643–648.

[164] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, "Performance Comparison of IEEE 802.1 TSN Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS)", *IEEE Access*, vol. 7, pp. 44 165–44 181, 2019.

[165] S. Kehrer, O. Kleineberg, and D. Heffernan, "A comparison of fault-tolerance concepts for IEEE 802.1 Time Sensitive Networks (TSN)", in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, IEEE, 2014, pp. 1–8.

[166] S. Qian, F. Luo, and J. Xu, "An Analysis of Frame Replication and Elimination for Time-Sensitive Networking", in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, ACM, 2017, pp. 166–170.

[167] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Reliability Analysis of TSN Networks Under SEU Induced Soft Error Using Model Checking", in *2019 IEEE Latin American Test Symposium (LATS)*, IEEE, 2019, pp. 1–6.

[168] F. Prinz, M. Schoeffler, A. Lechler, and A. Verl, "End-to-end redundancy between real-time I4. 0 Components based on Time-Sensitive Networking", in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, vol. 1, 2018, pp. 1083–1086.

[169] F. Prinz, M. Schoeffler, A. Lechler, and A. Verl, "Dynamic real-time orchestration of I4. 0 components based on time-sensitive networking", *Procedia CIRP*, vol. 72, pp. 910–915, 2018.

[170] I. Alvarez, J. Proenza, and M. Barranco, "Mixing Time and Spatial Redundancy Over Time Sensitive Networking.", in *DSN Workshops*, 2018, pp. 63–64.

[171] L. Lamport, "Time, clocks, and the ordering of events in a distributed system", *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[172] J. L. Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization", *Information and computation*, vol. 77, no. 1, pp. 1–36, 1988.

[173] L. Lamport and P. M. Melliar-Smith, "Byzantine clock synchronization", in *Proceedings of the third annual ACM symposium on Principles of distributed computing*, Citeseer, 1984, pp. 68–74.

[174] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems", *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 933–940, 1987.

[175] K. B. Stanton, "Distributing deterministic, accurate time for tightly coordinated network and software applications: IEEE 802.1 AS, the TSN profile of PTP", *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 34–40, 2018.

[176] H. Guo, "Time Synchronization and Communication Network Redundancy for Power Network Automation", PhD thesis, The University of Manchester (United Kingdom), 2017.

[177] W. Wallner, A. Wasicek, and R. Grosu, "A simulation framework for IEEE 1588", in *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, IEEE, 2016, pp. 1–6.

[178] G. M. Garner and H. Ryu, "Synchronization of Audio/Video Bridging Networks using IEEE 802.1 AS", *IEEE Communications Magazine*, vol. 49, no. 2, pp. 140–147, 2011.

[179] R. Exel, T. Bigler, and T. Sauter, "Asymmetry Mitigation in IEEE 802.3 Ethernet for High-Accuracy Clock Synchronization", *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 3, pp. 729–736, 2013.

[180] P. Loschmidt, *On enhanced clock synchronization performance through dedicated ethernet hardware support*. na, 2010.

[181] P. Loschmidt, R. Exel, and G. Gaderer, "Highly accurate timestamping for ethernet-based clock synchronization", *Journal of Computer Networks and Communications*, vol. 2012, 2012.

[182] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "Synchronization quality of IEEE 802.1 AS in large-scale industrial automation networks", in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2017, pp. 273–282.

[183] C. Le and D. Qiao, "Evaluation of Real-Time Ethernet with Time Synchronization and Time-Aware Shaper Using OMNeT++", in *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, IEEE, 2019, pp. 161–164.

[184] S. Nsaibi, L. Leurs, and H. D. Schotten, "Formal and simulation-based timing analysis of Industrial-Ethernet sercos III over TSN", in *Proceedings of the 21st International Symposium on Distributed Simulation and Real Time Applications*, IEEE Press, 2017, pp. 83–90.

[185] J. L. Du and M. Herlich, "Software-defined Networking for Real-time Ethernet.", in *ICINCO (2)*, 2016, pp. 584–589.

[186] A. Gopalakrishnan, *Applications of software defined networks in industrial automation*, 2014.

[187] G. Kálmán, "Applicability of software defined networking in industrial ethernet", in *2014 22nd Telecommunications Forum Telfor (TELFOR)*, IEEE, 2014, pp. 340–343.

[188] M. Herlich, J. L. Du, F. Schörghofer, and P. Dorfinger, "Proof-of-concept for a software-defined real-time ethernet", in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2016, pp. 1–4.

[189] D Bruckner, R Blair, M Stanica, A Ademaj, W Skeffington, D Kutscher, S Schriegel, R Wilmes, K Wachswender, L Leurs, *et al.*, "OPC UA TSN-A new Solution for Industrial Communication", *Whitepaper. Shaper Group*, 2018.

[190] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN networks", in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2017, pp. 1–8.

[191] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog computing for industrial automation through time-sensitive networking (TSN)", *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 55–61, 2018.

[192] S. B. H. Said, Q. H. Truong, and M. Boc, "SDN-based configuration solution for IEEE 802.1 time sensitive networking (TSN)", *ACM SIGBED Review*, vol. 16, no. 1, pp. 27–32, 2019.

[193] T. Issariyakul and E. Hossain, "Introduction to network simulator 2 (NS2)", in *Introduction to network simulator NS2*, Springer, 2009, pp. 1–18.

[194] "OMNeT++ Tutorials", in *https://docs.omnetpp.org/tutorials/tictoc/*, 2019.

[195] J. Pan and R. Jain, "A survey of network simulation tools: Current status and future developments", *Email: jp10@ cse. wustl. edu*, vol. 2, no. 4, p. 45, 2008.

[196] M. Abuteir and R. Obermaisser, "Simulation environment for time-triggered ethernet", in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, IEEE, 2013, pp. 642–648.

[197] "The Secure Shell (SSH) Protocol Assigned Numbers", in *Network Working Group*, NWG, 2006.

[198] Y. Quan and G. Liu, "Drifting clock model for network simulation in time synchronization", in *2008 3rd International Conference on Innovative Computing Information and Control*, IEEE, 2008, pp. 385–385.