FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

TSN Demonstrator

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Christoph Lehr
Matrikelnummer 01525189

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof.Dr. Wolfang Kastner
Mitwirkung: Dipl.-Ing.(FH) Dieter Etz, MBA
Dipl.-Ing. Thomas Frühwirth

Wien, 11. Jänner 2020

Christoph Lehr	Wolfang Kastner

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

TSN Demonstrator

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Christoph Lehr
Registration Number 01525189

to the Faculty of Informatics

at the TU Wien

Advisor:	Ao.Univ.Prof.Dr. Wolfang Kastner
Assistance:	Dipl.-Ing.(FH) Dieter Etz, MBA
	Dipl.-Ing. Thomas Frühwirth

Vienna, 11th January, 2020

Christoph Lehr

Wolfang Kastner

Erklärung zur Verfassung der Arbeit

Christoph Lehr
Czerninplatz 4/15, 1020 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 11. Jänner 2020

Christoph Lehr

Acknowledgements

First and foremost I have to thank Dipl. Ing. Thomas Frühwirth for his assistance during each step of the thesis. Without his involvement, the project would have never been accomplished. I also want thank Ao. Univ. Prof. Dr Wolfgang Kastner and Dipl. Ing.(FH) Dieter Etz MBA. for providing the opportunity to work on this topic.

I would also like to thank Auguste-Marie Bamogo and Felix Ring from the company TTTech, who provided support material, which was essential to complete the project.

Kurzfassung

Ziel dieser Arbeit ist es, den aktuellen Stand des Time-Sensitive Networking (TSN) Standards zu evaluieren und ein Demonstrationsnetzwerk aufzubauen. Zunächst wird der erforderliche technische Hintergrund dargestellt. Darauf folgen eine Einführung in allgemeine Themen von Time-Sensitive Networking und die dazugehörigen Standards.

Um den aktuellen Stand der auf dem Markt verfügbaren Produkte zu ermitteln, wurde eine Umfrage bei einer Reihe von Herstellern durchgeführt. Die Hauptgruppen der berücksichtigten Produkte, sind Chips und Komplettsysteme.

Zwei der zuvor bewerteten Geräte werden zum Aufbau eines Demonstrationsnetzwerks verwendet. Das Netzwerk zeigt, wie die Konfiguration abläuft und wie die Interoperabilität zwischen verschiedenen Herstellern funktioniert. Um die Echtzeitfähigkeiten von Time-Sensitive Networking (TSN) zu demonstrieren, wird ein unidirektionaler Datenfluss auf einem stark ausgelasteten Netzwerk simuliert. Um den TSN-Datenfluss auszuwerten, wird die Netzwerkverzögerung mit einem vergleichbaren Best Effort (BE)-Datenfluss verglichen.

Abstract

The goal of this thesis is to evaluate the current state of Time-Sensitive Networking (TSN) related standards and to build a demonstration network. First, the required technical background, an introduction to general topics in TSN and the contributing standards are provided.

To show what the current state of available products on the market is, a survey on a set of manufacturers was conducted. The main groups taken into account are chips and devices.

A set of the prior evaluated hardware is than used to build a demonstration network. The network is used to show how configuration works and how interoperability between manufacturers is solved. To demonstrate the real-time capabilities of TSN, a unidirectional data flow is configured and a high load is applied to the network. To evaluate the TSN data flow, the network delay is compared to Best Effort (BE) traffic.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

The emergence of Internet of Things (IoT) led to a growing number of connected devices, not only in households, but also within industrial applications. With Industrial Internet of Things (IIoT), more data is collected in industrial automation. Common field buses like Controller Area Network (CAN) or ProfiBus typically used in these applications lack the required bandwidth. Modern cars, with Advanced Driver Assistance Systems (ADAS) and Autonomous Driving (AD) systems, have an ever growing amount of sensor data. Therefore, Ethernet-based protocols were developed to keep up with the bandwidth demands. One of these solutions is Time-Sensitive Networking (TSN).

Ethernet is a widely used standard, especially in Information Technology (IT). Some of the benefits are the comparatively low prices of network hardware and the easy extensibility. Modern Ethernet devices are capable of transport speeds of several Gigabit per second (Gbit/s), this exceeds the capabilities of any available field bus. Since TSN is based on Ethernet, high priority sensor data can be transported on the same network as user data. This is often called the convergence between IT and Operational Technology (OT).

One of the benefits of TSN being an open protocol is that there is no single vendor which controls the specification. This is often the case with proprietary protocols. In general, that does not need to be a drawback, specifications controlled by a single company are often quite stable. Nonetheless, requirements on applications change and the protocol may need updates. In standardisation organisations like Institute of Electrical and Electronics Engineers (IEEE), members can cooperate to steer the direction of the standard. This typically leads to quite versatile protocols and broad usage, since a lot of big companies work together. Through open standards any supplier of hardware or software, can, if desired, produce tools and devices for this standard. In contrary to open standards, the sole owner of the specification of proprietary protocols decides who is allowed to provide tooling or devices. This can lead to a single vendor lock, where everything is to be obtained through a single source.

Since TSN is a rather new protocol, building a demonstration network is rather useful. It helps understand the current state and the available functionality in a controlled environment. In certain circumstances, it helps to evaluate what is currently provided by different manufacturers. It is especially useful to test out how each device can be configured and used in different set-ups. A not negligible point is, how the configuration tools work, if there is already something available for the selected platforms. At least currently there is some scope for interpretation and not all devices work with each other.

1.2 Aim of the work

The goal of this thesis is to evaluate the current state of the TSN related standards and build a demonstration network. Main points in building the demonstrator are: conception, configuration and setting it up with components of at least two manufacturers. To demonstrate the real-time capabilities of TSN, a unidirectional data flow is configured and a high load is applied to the network.

1.3 Methodological Approach and Structure of the Work

To provide the required technical background, an introduction to general topics in TSN is given. For more profound information, a shallow dive into the standards and amendments contributing to TSN is provided. To understand the domains where TSN shall be applied, a set of use-cases was composed. This is the content of Chapter 2.

To show what is the current state of available products on the market, a survey on a set of manufacturers was conducted. The main groups taken into account are chips and devices. The results are presented in Chapter 3 .

A set of the prior evaluated hardware is then used to build a demonstration network. The network is used to show how configuration works and how applications can benefit from TSN. The detailed description for the demonstrator can be found in Chapter 4.

In Chapter 5, a final statement to the gathered results and findings is given.

CHAPTER 2

Technical background

2.1 Introduction to Time-Sensitive Networking

Time-Sensitive Networking (TSN) is currently developed by the IEEE association. TSN is not a single standalone standard, but a collection of several extensions to IEEE 802.1 standards. The idea behind TSN is to provide deterministic services through Ethernet-based networks. In addition to IEEE, organisations like Avnu Alliance and Industrial Internet Consortium (IIC) drive standardisation and adoption of this technology.

In order for TSN to support deterministic services, it uses generalized Precision Time Protocol (gPTP) for time synchronisation, see Section 2.1.1 for more details. Based on a synchronised timebase, Traffic Scheduling handles when which frame is allowed to be sent. For more details, see Section 2.1.2.

Since a TSN network is not omniscient, the single time-aware bridges and time-aware end stations need a configuration. The basic concept behind this configuration is the principle that a data-stream is transmitted from one talker to one or more listeners. A *stream* is a a unidirectional flow of data (e.g. audio and/or video) from one end-point to one or more end-points. A talker is the source or producer of a stream and a listener is the receiver or the consumer of a stream. The configuration is exchanged over the User/Network Interface (UNI). In general, talker provide requirements for data-streams and the network checks if these can be fulfilled. The Stream Reservation Protocol (SRP) is used to establish the reservation of resources across a network for a stream in order to fulfil its requirements. For more details on this, see Section 2.2.2.

2.1.1 Time Synchronisation

Each computer has some kind of clock integrated. Most of these clocks are cheap and inaccurate. The drift is an indicator how accurate the used clock is, or in other words, how far the clock “drifts” away in a certain amount of time, e.g. clocks in commercial off-the-shelf hardware can drift multiple seconds over a single day. In systems with (hard) real-time requirements, this can lead to serious problems. Many systems require precise clock synchronisation, since time is the only reference available to determine the correct order of executing different processes.

Time or clock synchronisation is used to coordinate the single clocks in a system. Since systems not only consist of a single piece of hardware, but are often connected via communication buses, protocols were developed to serve the need of synchronisation. How this works can be separated into two groups: external and internal synchronisation. External synchronisation uses a reference like Global Positioning System (GPS) with very precise clocks to synchronise the network. Internal synchronisation just synchronises the resources inside a single network, but does not use any outside reference. Commonly known protocols are Network Time Protocol (NTP) for external synchronisation and Precision Time Protocol (PTP) for internal synchronisation.

NTP uses a Client-Server Architecture to exchange time information. The server is synchronised to a hardware clock, e.g. GPS. The connected network is separated into different hierarchy levels called *stratum*, which indicate how far away the time source is. For example, hosts in stratum 1 are directly connected to the time source. Due to path delays in the network the higher the stratum, the more accuracy gets lost. A big advantage of NTP is its widespread use and the availability of free NTP-servers on the Internet.

PTP is used to get accuracy in the sub-microsecond area. To achieve this, PTP uses frequency and phase correction mechanisms on the clock of the slave. The protocol uses a decentralized algorithm to synchronise the different clocks. Each PTP network contains a master clock and a set of slaves, the master clock is typically referred to as the *grandmaster*. Each member waits to receive a so-called announce message. This message contains, along with other fields, the quality of the clock of the sender. If the quality of the clock is higher than the clock the node is currently synchronised to, the node synchronises to the better clock. To synchronise the clocks on these levels, special hardware support is needed. In Section 2.2.1, a more detailed description of PTP is provided.

2.1.2 Traffic Control and Scheduling

Traffic scheduling is a part of Media Access Control (MAC), which handles how each client accesses a physical medium. There are a lot of ways how to control the access to a network. Most networks use BE scheduling and Carrier-Sense Multiple Access (CSMA) algorithms. BE means that no real-time guarantees can be provided for the traffic that is transported by the network.

In communication buses like CAN, Carrier-Sense Multiple Access - Colission Resolution (CSMA-CR) is used. This protocol tries to avoid collisions entirely by introducing identifiers and using bit arbitration. When a transmission is started, the sender reads every bit which is transmitted and compares them to the current state of the bus. If a bit of the identifier differs, the sender stops transmitting. When the whole identifier is transmitted, only one sender, the one with the highest priority, remains. Mechanisms like these are called (bit) arbitration. For instance, CAN uses 0 as the dominant state, which means the message with the lowest identifier has the highest priority. In Ethernet-based networks Carrier-Sense Multiple Access - Colission Detection (CSMA-CD) is used. Like in CSMA-CR, the sender tries to transmit a message and checks what bit is actually on the bus. If the bit differs, a collision has happened and both senders stop transmitting. After an arbitrary amount of time, the host retries to access the communication medium.

For regular networks, these methods are sufficient, but for real-time applications too many collisions may occur on CSMA-CD networks. This improved by introducing switches to Ethernet, nonetheless it is not sufficient for real-time traffic. Time-Division Multiple Access (TDMA) is a multiple access scheme, which is able to use the whole bandwidth provided by the channel. The access to the bus is separated into time slices, where an application is only allowed to access the bus during its period. If the underlying network has switching capabilities, the access can be optimized to allow network traffic on different network segments at the same time. Therefore, the bridges inside the network have to filter out unwanted traffic of sub-networks to avoid collisions. For TDMA to work reliable, time synchronisation is essential.

To overcome the limitations of BE, traffic shaping and policing were introduced. Traffic policing does network monitoring and enforces so-called traffic contracts. A traffic contract is a contract between the network and an application. The application informs the network what kind of data is transmitted and which requirements for the communication exist. Traffic shaping manages the bandwidth of the network traffic by slowing certain packets down and putting other packets faster through the queues.

Network scheduling goes one step further and defines the time at which a client is allowed to send. Therefore, a schedule for the complete network is created, whereby information from traffic policing and traffic shaping is used as an input. The access to the schedule can be done on various levels, either directly on the host or inside a bridge or switch. In both cases, packages are buffered/queued until a given timeslot begins. Then the queue(s) will be opened and start sending the queued data. The two ideas behind network scheduling are to reduce the number of collisions and to reduce the latency in a network. There are multiple models used, e.g. when using Time-Triggered Ethernet (TTEthernet) the schedule is created on frame level, in TSN frames are grouped into traffic classes.

2.1.3 Network Configuration

Network configuration is a very broad term. In general it is about configuring devices in a network. This starts at the lower levels with switches and routers, to servers providing Dynamic Host Configuration Protocol (DHCP) and Domain Name System (DNS). In terms of the OSI-model this affects the layers 2 to 4. In more general terms, network configuration is used to handle which Internet Protocol (IP) addresses are provided, which proxy servers shall be used and so on. To accomplish this, protocols like Simple

Network Management Protocol (SNMP) and Network Configuration (NETCONF) were developed.

SNMP is a widely used protocol and newer versions were developed to provide performance flexibility improvements as well as a security extension. In general, the protocol uses a so-called Management Information Base (MIB). There the system is described and the current status is stored. By exposing these values, SNMP provides useful network data to network administrators. For some entries in the MIB, it is also possible to remotely manipulate them. SNMP works as follows: a *manager* or a group of *managers* monitors a set of *agents*. On a device, also called *managed device*, one or more agents can run. An agent can monitor the whole system or a set of applications running on a system.

NETCONF was developed as a successor to SNMP, since the intended feature to modify devices was barely used. Via Remote Procedure Calls (RPCs), NETCONF exchanges XML encoded messages. In contrast to SNMP, NETCONF is executed on top of secure transport protocols like SSH. Typically, a YANG-model is used to define the capabilities of a device.

2.2 Relevant Standards for TSN

Since TSN consists of a variety of IEEE standards, this section contains the standards on which TSN is founded. The following subsections cover the content of all the standards which contribute to TSN, a short overview is listed in Table 2.1. At the moment not all listed amendments are already incorporated in the latests releases of the several standards.

Short-Title	Content	Citation
IEEE 802.1AS 2011	Time Synchronization	[80211]
IEEE 802.1Qcc 2018	SRP Enhancements	[80218c]
IEEE 802.1CB 2017	Frame Replication and Elimination	[80217c]
IEEE 802.1Qca 2015	Path Control and Reservation	[80216b]
IEEE 802.1Qci 2017	Filtering and Policing	[80217a]
IEEE 802.1Qbv 2015	Scheduled Traffic	[80216c]
IEEE 802.1Qch 2107	Cyclic Queuing and Forwarding	[80217b]
IEEE 802.1Qbu 2016	Frame Preemption	[80216a]

Table 2.1: List of Relevant IEEE Standards and Amendment for TSN

2.2.1 802.1AS: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks

As described in Section 2.1.1, one of the crucial elements of TSN is time synchronisation, which is covered by the IEEE 802.1AS-2011 standard [80211]. The main element described here is the generalized Precision Time Protocol (gPTP), which is used to guarantee a synchronised timebase across the network. gPTP is a specialised subset of the PTP, and intended to be used in a bridged packet-switched Local Area Network (LAN). To understand how the time synchronisation procedure works, some definitions are needed, see [80211]:

time-aware bridged LAN: a set of time-aware systems which are connected by a LAN which supports gPTP.

gPTP domain: a connected subset of a *time-aware bridged LAN*.

grandmaster: is the master clock of a whole network, any time-aware system can be a *grandmaster*. The Best Master Clock Algorithm (BMCA) ensures that all time-aware systems in a gPTP domain use the same grandmaster.

time-aware bridge: receives time information; if it is not the grandmaster it applies corrections to the LAN and the bridge itself. The corrected information is retransmitted to the connected nodes.

time-aware end station: if it's not the grandmaster, it is a recipient of time information.

time-aware system: is either a time-aware bridge or a time-aware end station.

The main components of gPTP networks are illustrated in Figure 2.1. In this figure, the master-slave concept of the clocks is depicted. A *master port* is the time reference for a *slave port*. A slave port synchronises the local clock to the network. In contrast, a *passive port* receives the timing information and does no further processing. If a network path is interrupted, the passive port can become a slave port to maintain synchronisation.

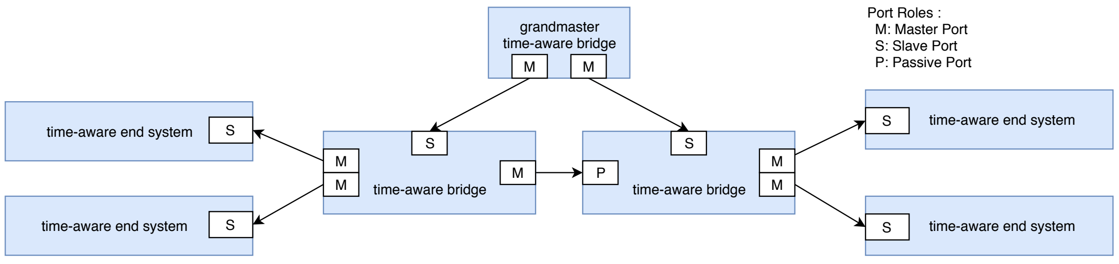

Figure 2.1: Example hierarchy of a time-aware network.

In each time-sensitive network, which uses gPTP, a grandmaster exists. The grandmaster is a time-aware system on its own and provides the timebase for the network. To synchronise the network clocks with its time, the grandmaster transmits its local time to all directly attached devices. All time-aware systems use the received time, add the propagation delay and synchronise their local clocks to the corrected time. If the time-aware system is a time-aware bridge, the adapted time will be transmitted to all its directly connected devices. This step is repeated through the network until all time-aware systems are synchronised to the grandmaster's clock.

If not set in advance, the grandmaster can be selected dynamically by utilising the BMCA. Every member of the network can attend the selection of the new grandmaster. This algorithm runs on each time-aware system independently. Each node, including the sender of the announcement, compares the received announcement to the used master clock. If the result indicates that another clock has a better quality, the time-aware system announces the result to the other nodes. When the current grandmaster receives this messages and detects that the received clock is better, the grandmaster stops sending time information and the new sender with the better clock is the new grandmaster. The result of this process is a time-synchronisation spanning tree with the grandmaster as root.

In all networks, each message has at least a minimal delay to traverse a network, each time-aware system has to evaluate the propagation delay between itself and its directly attached devices. There are many different algorithms for this delay calculation, but most of them are based on the same principle. The measurement is based on the assumption that the path delay for sending and receiving a message is equal in both directions. The sequence of the message exchange is depicted in Figure 2.2, the steps are as follows:

1. The master sends a sync message to a slave at point *T1*, the reception time is stored as *T2*

2. The master sends a follow up message containing the timestamp *T1*

3. The slave sends a delay request message to the master and stores the time of sending as *T3*

4. The master sends a delay response with the reception time of the frame, which is stored as *T4*

5. The adjustment of the slave clock is calculated as follows:

$$adj = -\frac{(T2 - T1) - (T4 - T3)}{2}$$

IEEE is currently working on a new revision of 802.1AS which will include several extensions designed for TSN. One of these features is the introduction of redundant clocks in the network. However, at the time of writing, the new revision has not yet been published.

[1]`https://doc.dpdk.org/guides/sample_app_ug/ptpclient.html` accessed 07.09.2019

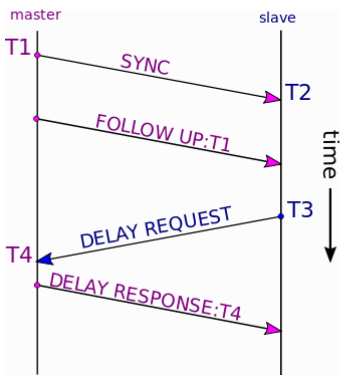

Figure 2.2: PTP delay measurement process.[1]

2.2.2 802.1Qcc: Stream Reservation

With the amendment 802.1Qcc „Stream Reservation Protocol (SRP) Enhancements and Performance Improvements“ [80218c], the means of configuring TSN networks, will be added to the IEEE 802.1Q standard. To understand the content of this section, some definitions are required.

In order to transmit a configuration across a bridged network the User/Network Interface (UNI) is introduced, which provides two interfaces: the user and the network interface. The user interface represents talkers and listeners in a network. It is used to transmit requirements of a data-stream, without knowledge of the underlying network. The second interface, the so-called network interface, represents the bridges. This interface obtains the requirements and analyses the network to evaluate, if the requirements can be fulfilled or not.

The TSN User/Network Configuration Information (UNCI) is not bound to a specific protocol, for instance SRP can be employed for this use case. In practice, any protocol which supports UNCI shall integrate the three high level classes, which are defined as follows [80218c]:

UNCI-talker: elements from user to network which specify a talker for a single stream

UNCI-listener: elements from user to network which specify a listener for a single stream

status: elements from network to user that specify the status of the stream's network configuration for a talker or listener. This group notifies the user when the stream is ready for use (or a failure occurred).

Since these definitions of UNCI-talker and UNCI-listener overlay with the already existing ones, the prefix UNCI was added.

When using the SRP, UNCI-talkers provide the necessary stream characteristics to the bridges, in order to reserve the required resources for the stream. UNCI-listeners provide the necessary stream characteristics for the reception of a stream. These attributes are matched by the bridges via the StreamID, which is part of all attributes. Bridges along the way, from UNCI-talker to UNCI-listener, process this information, adjust it if needed, and forward it as Multiple Stream Registration Protocol (MSRP) attributes to the directly connected bridges.

In general, the following three models for a network configuration are provided by this amendment:

fully distributed model: the network is configured in a way that all end stations communicate the requirements directly over the TSN network.

centralized network/distributed user model: the configuration is directed to/from a Centralized Network Configuration (CNC) entity, which holds all stream requirements and the bridge configuration is done by the CNC entity.

fully centralized model: a Centralized User Configuration (CUC) entity discovers all the end stations in a network and retrieves the capabilities and requirements of them. To configure the underlying network, the CUC exchanges its data with the CNC.

The fully centralized model is often available in two variants available, the above described variant is often called “online”. To second variant is called “offline”. The difference is that in a central unit the whole network schedule is created a-priori and no exchange of requirements is needed. This variant has two crucial advantages, the feasibility of the schedule can be checked in advance and the needed computational power of the network devices can be reduced.

2.2.3 802.1CB: Frame Replication and Elimination for Reliability

The IEEE 802.1CB standard [80217c] handles the identification of streams, the replication and elimination of packets in order to increase the reliability of transmissions.

The Frame Replication and Elimination for Reliability (FRER) was introduced to increase the reliability of streams, in particular to reduce package loss. To achieve this, packages get a sequence number and are replicated by the talker itself or a relay system on the way from a talker to a listener. The replicated messages are eliminated in the destination end system or on a relay system. The standard has been defined in a way such that FRER can also be used if it is supported only by the end systems, but not by the network infrastructure (switches). In this case, frames are only replicated at the sender and duplicates are eliminated at the receiver.

In Figure 2.3, an example of FRER is depicted. The leftmost box is the talker. It encodes a sequence counter into the package and transmits this package to both of its directly connected relay systems. In the relay systems, sequence recovery functions eliminate the duplicates and the non-duplicates are copied into a new member stream with the same sequence counter. At the listener, illustrated as the block on the rightmost side, the two member streams are now combined and the duplicates get eliminated. In this particular network all seven one-link failures and 16 out of 21 possible two-link failures can be tolerated.

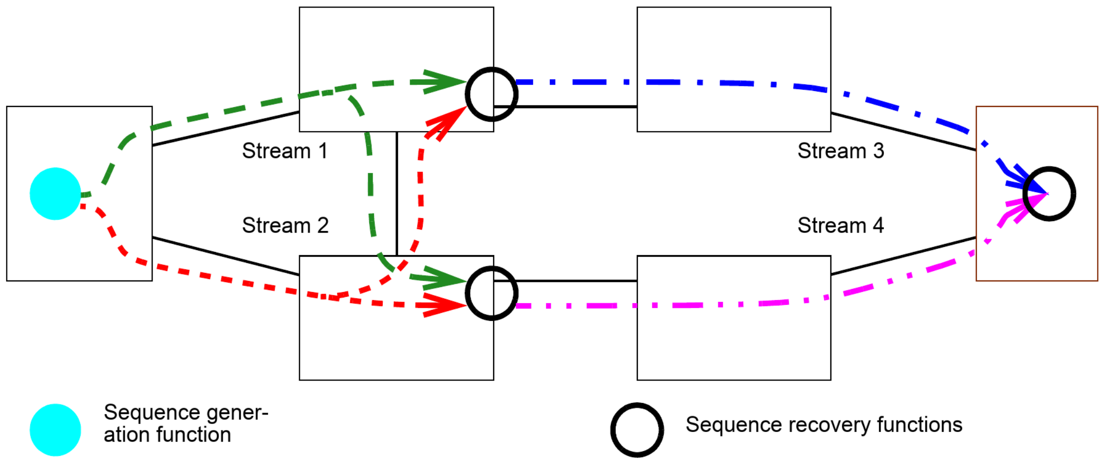

Figure 2.3: Frame Replication Elimination Example [80217c]

2.2.4 802.1Qca: Path Control and Reservation

To give a deeper understanding what amendment 802.1Qca stands for, an introduction to Intermediate System to Intermediate System (IS-IS) and Open Shortest Path First (OSPF) is needed. These protocols are used for routing inside a single routing domain, but the main purpose is to determine the shortest routes through networks. Typically, this is done by routers and layer 3-switches. Additional information is added to the MAC-Header directly on layer 2 of the OSI-model. One of these routing mechanisms is Shortest Path Bridging (SPB) which interoperates with different Spanning Tree Protocols (STPs).

The Path Control and Reservation (PCR) is an extension to IS-IS protocols, and depends on SPB. The STPs create a logical tree from the network topology. The main idea behind this, is to prevent bridge-loops in a network. Since this is done during runtime, if a connection breaks, the network routes can be reconfigured to use the redundant connection instead. SPB does extend STP by also utilising these redundant connections, which increases the throughput of a network. Therefore, each bridge maintains a local shortest path tree for itself.

PCR was introduced to provide a deterministic routing mechanism for unicast and multicast traffic. Therefore, so-called Path Computation Elements (PCEs), are introduced. Each PCE determines paths for point-to-point, point-to-multipoint and multipoint-to-multipoint connections. PCR provides the topology of the network for other services, e.g. in a database. Protocols like MSRP can use this information, to reserve the network resources for the streams and FRER can use the information for handling the redundant paths.

2.2.5 802.1Qci: Per-Stream Filtering and Policing

Bridges, as one of the fundamental parts of a network, relay the traffic to certain ports in order to reduce the traffic on the network. One part of relaying the traffic is filtering. This can be done based on the MAC address and the VLAN membership of the frame.

With the IEEE 802.1Qci-2017 amendment, [80217a] Per-Stream Filtering and Policing (PSFP) was introduced. This amendment adds support for filtering streams on relay systems. The forwarding process consists of multiple stages, depicted Figure 2.4. PSFP is a member of the „Flow Metering“ stage.

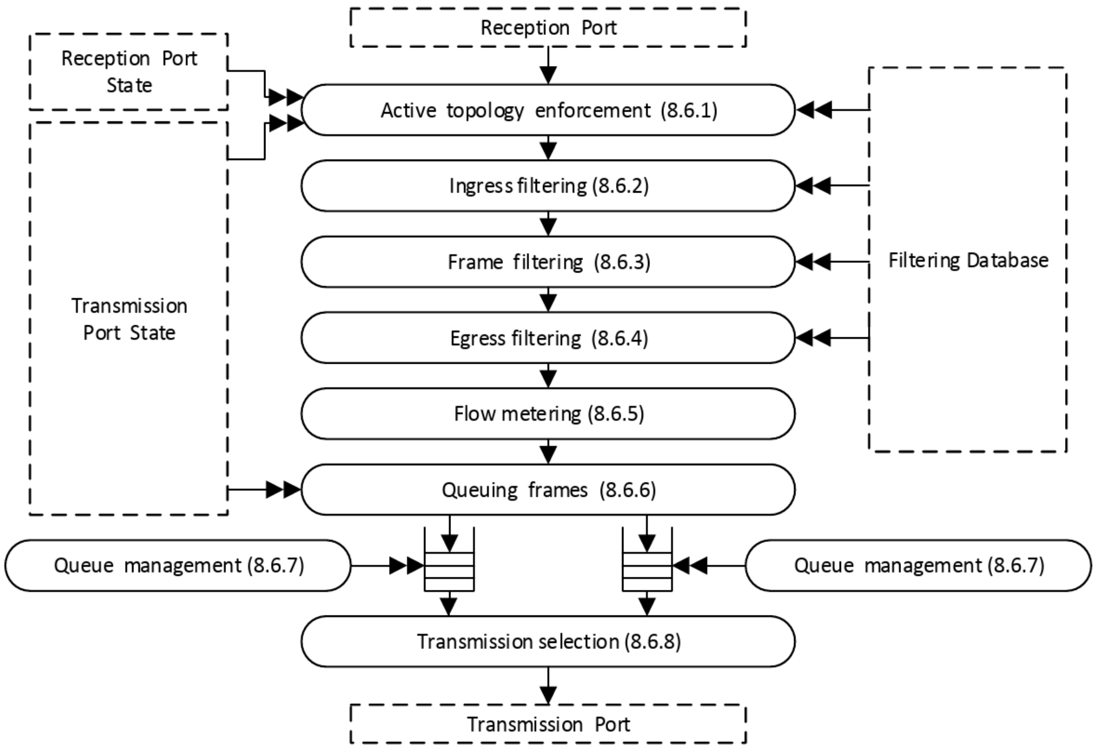

Figure 2.4: Forwarding Process [80218a]

Filtering of frames is split up into the following three tables:

stream filter instance table: a stream filter determines what filtering and policing actions are taken for packets of a certain stream. Another parameter of this table is a link to a stream gate instance.

stream gate instance table: a stream gate instance holds the gate state, which is either open or closed. If the gate is open, frames are allowed to pass through the gate. Another important parameter is the Internal Priority Value (IPV) of the stream. The IPV is used to determine the correct traffic class.

flow meter instance table: this table holds all parameters needed for the bandwidth profile parameters and algorithm defined in MEF Specification 10.3 [MEF13]

After filtering, each frame is inserted into a queue. The frames which shall be sent over the network are categorised into traffic classes by PSFP and the IPV. Thereby, each traffic class corresponds to a queue.

2.2.6 802.1Qbv Enhancements for Scheduled Traffic

In combination with support for gPTP, in the IEEE 802.1Qbv-2015 amendment [80216c] additional support for scheduling traffic was added. This amendment adds traffic classes and queuing algorithms to the Ethernet standard.

Since one of the key features of TSN is getting reliable access to an Ethernet-based network, TDMA-based traffic scheduling is one the main aspects. The transmission selection algorithm determines, which gates shall be opened next. Each queue is associated with a separate transmission gate, which allows to transmit frames based on a known time-scale. This information is stored in the gate control list. If the transmission gate is open, frames of the corresponding queue are selected for transmission (cf. Figure 2.5). If enhancements for scheduled traffic are not supported, every gate is assumed to be open.

Scheduled traffic state machines were introduced for controlling the transmission gates:

cycle timer state machine: handles the execution of gate control lists and the compliance of the defined gate cycle times.

list execute state machine: handles the execution of the operations defined in the gate control lists and establishes the needed delays between operations.

list config state machine: handles updating the schedule and propagates the updates to the other two state machines.

2.2.7 802.1Qch: Cyclic Queuing and Forwarding

IEEE amendment 802.1Qch-2017 [80217b] adds „Cyclic queuing and forwarding (CQF)“ to guarantee a more deterministic behaviour of the forwarding process. It can be used for deterministic delivery and calculating the latency in a network. It shall be noted that this amendment mainly adds informative content.

The principle of CQF is rather straightforward: transmission is handled in a cyclic manner. This is done by dividing the time in intervals of length d. Frames are always

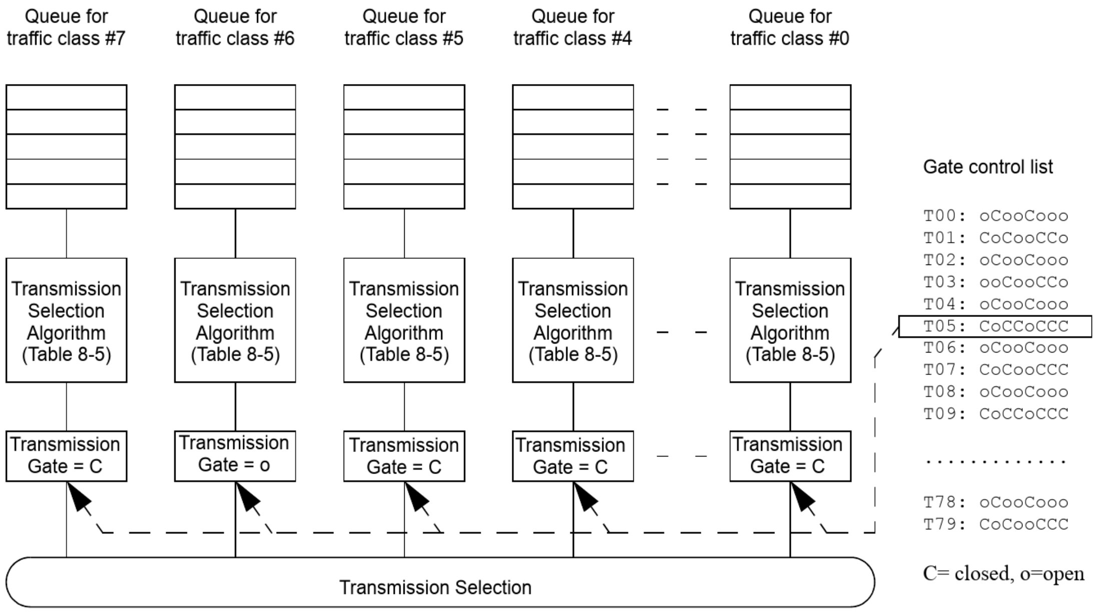

Figure 2.5: Transmission selection with gates [80218a]

transmitted during a single timer interval. At interval i, bridge A sends a frame to bridge B, which is received in the same time interval. Bridge B now forwards the frame to bridge C in the time interval $i+1$ and so on. This is done under the assumption that all devices connected to a bridge have the same interpretation for the start of the cycle i and the duration d. With these assumptions, the maximum delay is bounded to $(h+1)*d$, where h is the number of hops. In addition, the minimum delay is $(h-1)*d$. The amendment includes useful hints for an implementation.

2.3 Use-cases of TSN

This section addresses several use-cases of TSN.

2.3.1 Audio Video Bridging (AVB Systems)

Audio and video connections were historically only one-way analogue connections. Digital Audio-Video (AV) connections still had the limitation of a one-to-one connection. 802.1BA-2011 Audio Video Bridging (AVB) Systems were created to overcome the shortcomings of the known solutions. The main idea behind AVB is to provide real-time audio and video stream via Ethernet. AVB was designed to replace the complexity of point-to-point analogue cables and proprietary network solutions.

AVB provides the following four services

- Precise synchronisation, through gPTP

- Traffic shaping for media streams
- Admission controls
- Identification of non-participating devices.

TSN was founded on the idea and the knowledge inherited from AVB.

2.3.2 Mobile Fronthaul Networks

Another use-case is Fronthaul for cellular networks, which is used to connect the radio equipment to their remote controller. Specifically for this use-case, IEEE released a new standard – IEEE 802.1CM-2018 Time-Sensitive Networking for Fronthaul [80218b].

Fronthaul is a new type of Radio Access Network (RAN) architecture used in mobile networks. In this model each cell has its own Remote Radio Head (RRH) and a Baseband Unit (BBU) is located in a central location. The connection between the RRH and the BBU is realized via the Common Public Radio Interface (CPRI) protocol, which utilises optical connections to guarantee a high throughput.

Typically, Fronthaul networks are point-to-point networks. IEEE tries to provide a compelling alternative with bridged networks. Probably the biggest advantages are multipoint-to-multipoint and rooted-multipoint connectivity between endpoints in Fronthaul networks. A deterministic network that can handle the stringent Fronthaul requirements will provide benefits for the operators as well as the users of 4G and 5G networks.

2.3.3 Industrial Automation

Industrial Automation is one of the largest industries pushing TSN. With a reliable and open standard for an Ethernet-based network solution, they expect faster communication in comparison to field buses, as well as dropping prices. Some original field buses have real-time guarantees and deterministic behaviour. Off-the-shelf Ethernet components are not able not provide these capabilities. Another advantage is the easier extendibility and the straight forward compatibility with standard Ethernet hardware. This eases the maintenance of the network and helps connecting production systems to the enterprise level.

The International Electrotechnical Commission (IEC) is working together with IEEE to release a designated TSN Profile (IEC/IEEE 60802) specifically designed for industrial automation. Part of this profile are guidelines and a selection of IEEE 802 standards and features. The idea behind this is to support OT and other traffic in the same network.

2.3.4 Automotive networks

More and more powerful ADAS and AD systems are demanded in modern cars, therefore, communication networks with higher throughput are required. Since Peripheral Component Interconnect Express (PCIe) is only targeted for short distances and standard Ethernet is not reliable enough, a new solution is needed. For highly reliable fault-tolerant systems, TTEthernet exists. It provides a time-triggered architecture and is mostly used in aviation and space programs. In the automotive sector, using TTEthernet is mostly too expensive and/or too strict. The detailed specification of TTEthernet can be found as SAE AS6802 [AS616]. Since AVB made the first steps for camera systems inside, TSN is expected to provide the needed leverage for high bandwidth and high reliability inter-ECU communication inside a car.

CHAPTER 3

TSN Components

This chapter addresses the available and future products with support for AVB and TSN. It is separated into two groups. The first group addresses silicon manufacturers which provide Systems on a Chip (SoCs) and Microcontroller Units (MCUs). The second group addresses manufacturers, which do not just provide chips, but whole TSN-switches or TSN-end systems which use TSN for connectivity to other systems, like robots or Programmable Logic Controllers (PLCs).

3.1 TSN Chips

The focus of this section is rather on chips, than on an Electronic Control Unit (ECU) or a whole system. The parts listed in this section can be separated into SoCs with or without switching capacity.

3.1.1 Texas Instruments

The Sitara™ processor line up[1] from Texas instruments is an ARM®-based SoC targeting industrial automation applications. The Sitara™ processors support up to six Industrial Ethernet ports which can be used for TSN. With the up to four ARM® Cortex-A53 cores, this SoC provides sufficient processing capabilities for many industrial systems.

3.1.2 Microchip

EtherSynch® is Microchip's platform for deterministic and robust Ethernet communication[2]. This technology provides a hardware-based implementation for extensive traffic

[1] `https://www.ti.com/processors/sitara-arm/overview.html` accessed on 26.05.2019

[2] `https://www.microchip.com/design-centers/ethernet/ethernet-devices/technology/ethersynch` accessed on 26.05.2019

scheduling, synchronization and communication processing. Due to the implementation in hardware, the chips can reduce the overhead in the host Central Processing Unit (CPU). All Ethernet-switches using EtherSynch® are capable of TSN. Microchip also provides the SAM V 71MCUs[3] which support AVB on at least one port.

3.1.3 NXP

NXP has a series of automotive-grade Ethernet switching Integrated Circuits (ICs). The SJA1105[4] series fully supports AVB. The SJA1105TEL additionally supports TSN features like time-aware traffic [80216c] and per-stream policing [80217a]. This is the next step for NXP, since the SJA1105-Switches also provide TTEthernet for automotive networks. Additionally, a demonstration board[5] is available. The board uses a LS1021A-processor and a SJA1105 switch, whereby four Ethernet ports can be used as a switch, the fifth port is internally connected to the SoC. To promote their platform, NXP provides an open-source Linux distribution Open Industrial Linux (OpenIL)[6].

3.1.4 Broadcom

Broadcom provides two line-ups of switches with TSN functionality. the RoboSwitch™ and the StrataConnect®. As part of the RoboSwitch™ line the BCM53112[7] and the BCM53154[8] provide five Gigabit (Gbit)-Ethernet ports. The BCM53570[9] as part of the StrataConnect® line provides up to 48 Gbit-Ethernet ports with switching capacity up to 50 Gbit/s. The main targets for this chip are Industrial-Ethernet and 5G Fronthaul networkss. The SoCs are accompanied by an ARM® Cortex-M7 chip, which eases the design of industrial switches, since no additional CPU is needed.

3.1.5 Marvell

Marvell currently provides a small set of chips in their product line-up with support for AVB and TSN. The 88EA6321[10] and the 88Q5050[11] are targeting the automotive industry and provide seven and eight Ethernet ports. The 88Q5050 provides additional security features like a Deep Packet Inspection (DPI). A third chip, the 88E6341[12], provides 11 Ethernet ports with a speed up to 2.5 Gbit/s.

[3] `https://www.microchip.com/design-centers/32-bit/sam-32-bit-mcus/sam-v-mcus` accessed 26.05.2019

[4] `https://www.nxp.com/part/SJA1105TEL` accessed on 07.09.2019

[5] `https://www.nxp.com/part/LS1021ATSN-PA` accessed on 07.09.2019

[6] `https://www.openil.org/` accessed on 27.07.2019

[7] `https://www.broadcom.com/products/ethernet-connectivity/switching/roboswitch/bcm53112` accessed on 27.07.2019

[8] `https://www.broadcom.com/products/ethernet-connectivity/switching/roboswitch/bcm53154` accessed on 27.07.2019

[9] `https://www.broadcom.com/products/ethernet-connectivity/switching/strataconnect/bcm53570` accessed on 26.05.2019

[10] `https://www.marvell.com/documents/xlt1kjv25lhhmhngwqlc/` accessed 27.05.2019

[11] `https://www.marvell.com/documents/kfnykfeorttbylwgqbxe/` accessed 27.05.2019

[12] `https://www.marvell.com/documents/kfnykfeorttbylwgqbxe/` accessed 27.05.2019

3.1.6 Renesas

Renesas as one of the bigger suppliers for automotive chips currently has no product which actively supports AVB or TSN. In their TSN demonstration setup[13], the RH850 microprocessor family[14] was used. The upgrade to additionally support TSN does not seem to be a big issue since it already supports a list of Industrial Ethernet protocols.

3.1.7 Infineon

As one of the major chip providers for automotive ECUs, the Infineon Aurix^{TM} [15] line-up of MCUs, guarantees functional safety at the highest Automotive Safety Integrity Level (ASIL). The ISO 26262 defines measures for functional safety as well as Faults in Time (FIT) rates. 1 FIT corresponds to 1 activation of a fault every 10^9 hours of runtime. Devices and software, with the highest ASIL, i.e. ASIL-D, have to prove that they have a FIT-rate less then 10 FITs.

3.1.8 Xilinx

Xilinx is one of the leading manufacturers of Field-Programmable Gate Array (FPGA) technology. Currently, there is no dedicated TSN-Chip provided by Xilinx, but a subsystem which can be licensed[16].

3.2 TSN-Switches and End Systems

3.2.1 TTTech

The MFN100[17], which is an integral part of TTTech's Nerve platform, provides four Ethernet ports with support for TSN. TTTech does not use a SoC for the TSN-switch, but an FPGA[18]. The use of an FPGA brings the advantage of reprogrammable hardware, which is useful for later hardware upgrades. The used TSN-switch can be licensed from TTTech. The MFN100 combines TSN switching capabilities with additional features, e.g. a software-PLC. For evaluation purposes, TTTech also provides a PCIe-Expansion Card as development platform[19].

[13]http://renesasatces.com/renesas-ethernet-tsn-demonstrator/ accessed 27.05.2019

[14]https://www.renesas.com/sg/en/products/microcontrollers-microprocessors/rh850.html accessed 27.05.2019

[15]https://www.infineon.com/dgdl/Infineon-AURIX_Industrial-PP-v01_00-EN.pdf?fileId=5546d46269e1c019016a78da06b7548e accessed 27.07.2019

[16]https://www.xilinx.com/products/intellectual-property/1gtsn.html accessed 28.06.2019

[17]https://www.tttech.com/products/industrial/industrial-iot/nerve/ accessed on 26.05.2019

[18]https://www.tttech.com/wp-content/uploads/TTTech_DE-IP-Solution-Edge.pdf accessed on 28.07.2019

[19]https://www.tttech.com/products/industrial/deterministic-networking/hardware/de-evaluation-board/ accessed on 28.07.2019

3.2.2 Hirschmann/Belden

Hirschmann's BOBCAT Switches[20] provide up to 8 Ethernet ports which can provide up to 2.5 Gbit/s. The main selling point is a nearly free configurable interface to fit all connectivity demands.

3.2.3 Cisco

Cisco, as one of the largest providers of network equipment, provides via their Industrial Ethernet 4000 Series switches[21] a hardware platform with support for TSN. The switch supports up to 20 Ethernet ports and up to 16 Gbit-Ethernet ports. Cisco targets manufacturing sites, roadways and airports. Because of their known Cisco Internet Operating Systems, many network engineers are already trained to use their equipment. This gives an additional benefit as the time required to deploy the system is reduced.

3.2.4 Kontron

Currently, Kontron has one industrial-Personal Computer (PC), the KBox C-102[22], which is targeted on industrial automation with support for TSN. Kontron also supplies a TSN-Starterkit with its KBox and a TSN PCIe-expansion-card. Additionally, Kontron provides a set of modules, like the ESC1600-PTP and ESC2404-PTP[23]. These boards use a Broadcom BCM56440 chip and can be used as platforms for switches, but only AVB is supported.

3.2.5 Kuka

Kuka, as the second largest manufacturer of manufacturing robots in the world, is also interested in TSN[24]. The use of TSN was a step to create an open and interoperable architecture for their products and to make smart factories more efficient. Kuka will include TSN functionality in future products.

3.3 Comparison

The various products discussed so far are summarised in Table 3.1. The devices were separated into three device types: switches, end systems and switched end systems. Development platforms and demonstration boards are excluded from this table.

[20] http://www.hirschmann.com/de/Hirschmann/Industrial_Ethernet/OpenRail-Familie/BOBCAT/index.phtml accessed 27.05.2019

[21] https://www.cisco.com/c/en/us/products/switches/industrial-ethernet-4000-series-switches/index.html accessed 27.05.2019

[22] https://www.kontron.de/products/systems/embedded-box-pc/kbox-c-series/kbox-c-102.html accessed 28.07.2019

[23] https://www.kontron.de/products/iot/iot-industry-4.0/ethernet-tsn-switching/network-interfaces-tsn/esc1600-ptp-esc2404-ptp.html accessed 28.7.2019

[24] https://www.kuka.com/en-de/future-production/industrie-4-0/industry-4-0-digital-domains accessed 28.07.2019

Table 3.1: Available TSN hardware from various vendors

Manufacturer	Device	Device Type	No. Ports	comment
Analog Devices	FIDO5100	End system	2	
Analog Devices	FIDO5200	End system	2	
Broadcom	BCM53570	Switch	up to 48	Switching capacity up 50 Gbit/s
Cisco	4000 Series	Switch	up to 20	Switching capacity up to 16 Gbit/s
Hirschmann	BOBCAT	Switch	up to 8	Switching capacity up to 2.5 Gbit/s
Infineon	Aurix	End system	1	ASIL-D capabilities
Kontron	KBox C-102	Switched end system	up to 4	
Kontron	ESC1600-PTP	Switch	up to 24	only support for AVB
Kontron	ESC2404-PTP	Switch	up to 24	only support for AVB
Marvell	88EA6321	Switch	7	
Marvell	88Q5050	Switch	8	
Marvell	88E6341	Switch	11	Switching capacity up to 2.5 Gbit
Mircochip	SAM V 71®	End system	1	
NXP	SJA1105TEL	Switch	5	
Renesas	RH580	End system	1	only support for AVB
Texas Instruments	Sitara^{TM}	End system	up to 6	
TTTech	MFN100	Switched end system	4	

CHAPTER 4

Demonstration Network

To validate the capabilities of TSN networks, a small demonstrator was built. A set of different targets were used to show how configuration of such networks and devices is done, how to set them up and how interoperability works between different manufacturers. Two TTTech MFN100 switches and two NXP LS1021ATSN boards were connected to construct the demonstration network.

4.1 The Demonstrator

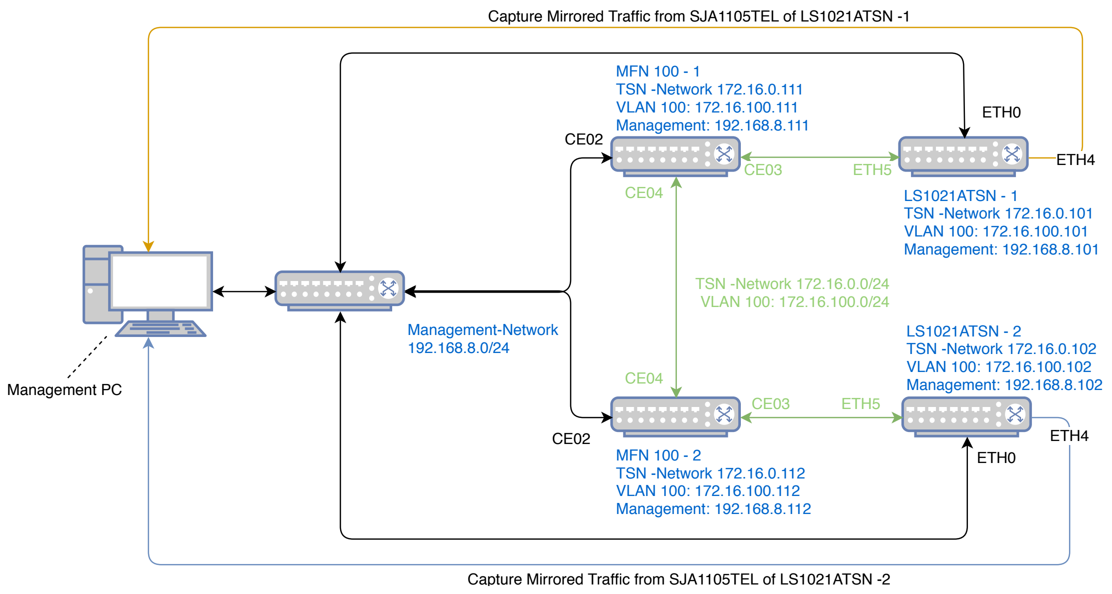
Capture Mirrored Traffic from SJA1105TEL of LS1021ATSN -1
MFN 100 - 1
TSN -Network 172.16.0.111
VLAN 100: 172.16.100.111
Management: 192.168.8.111
CE02
CE03
ETH5
ETH0
ETH4
CE04
LS1021ATSN - 1
TSN -Network 172.16.0.101
VLAN 100: 172.16.100.101
Management: 192.168.8.101
TSN -Network 172.16.0.0/24
VLAN 100: 172.16.100.0/24
Management-Network
192.168.8.0/24
Management PC
LS1021ATSN - 2
TSN -Network 172.16.0.102
VLAN 100: 172.16.100.102
Management: 192.168.8.102
CE04
CE03
ETH5
ETH4
CE02
ETH0
MFN 100 - 2
TSN -Network 172.16.0.112
VLAN 100: 172.16.100.112
Management: 192.168.8.112
Capture Mirrored Traffic from SJA1105TEL of LS1021ATSN -2

Figure 4.1: Network of TSN-Demonstrator

The demonstrator consists of two distinct networks, a management network and a TSN network. For the management network any router or switch can be used. The management network is used to distribute the configurations exported from SlateXNS to the different end systems. The second reason for using a dedicated management network is debugging of network and software issues. In Figure 4.1, a representation of the network is depicted. For the reliable data transmission, the VLAN 100 is used. For each of the LS1021ATSN boards, an additional connection to monitor the traffic of the built-in TSN-switch, was used. To do this, the switch was configured to mirror the traffic to a certain port.

4.1.1 Configuration Work Flow

The configuration procedure is depicted in Figure 4.2. Before starting to configure the actual network, a network model is required, in this case, a SlateXNS model is used. The first step is exporting the model in NETCONF. Steps 2a and 2b are executed only for the LS1021A and step 3 is executed only for the MFN100. In step 2a, the network schedule is extracted from the NETCONF files, which are exported from the SlateXNs model. In step 2b, the extracted schedule is used to create a configuration used by the SJA1105 switch. In step 3, the NETCONF file is transformed into a gate state list. In step 4, the files generated by steps 2 and 3 are deployed on the targets. Steps 5 and 6 use helper tools on the targets to configure the hardware. The final step is to set up the VLAN configuration, if not already done.

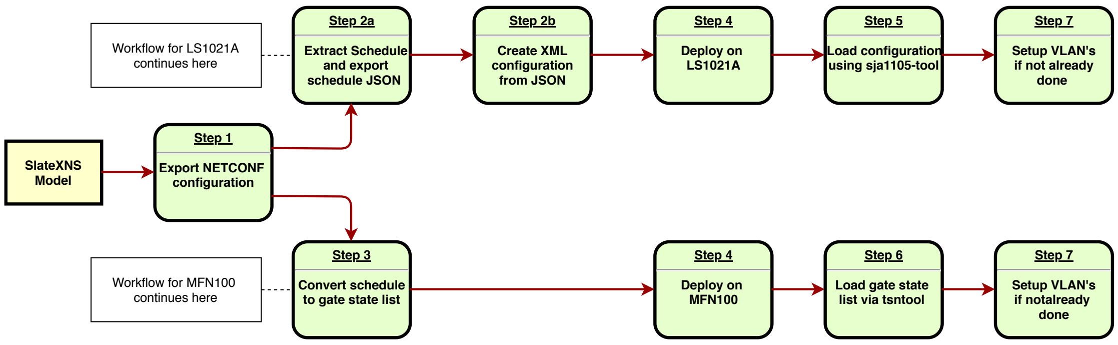
Workflow for LS1021A continues here
Step 2a
Extract Schedule and export schedule JSON
Step 2b
Create XML configuration from JSON
Step 4
Deploy on LS1021A
Step 5
Load configuration using sja1105-tool
Step 7
Setup VLAN's if not already done
SlateXNS Model
Step 1
Export NETCONF configuration
Workflow for MFN100 continues here
Step 3
Convert schedule to gate state list
Step 4
Deploy on MFN100
Step 6
Load gate state list via tsntool
Step 7
Setup VLAN's if notalready done

Figure 4.2: Configuration Work Flow

4.1.2 SlateXNS

SlateXNS is a tool provided by TTTech for modelling TSN networks. For smaller networks a graphical view is provided. For networks exceeding ten components or five streams, the textual view provides a better overview over the model. For fine-tuning of the model, this view also provides additional mechanisms, which impact the scheduling, e.g. how the single streams shall be mapped together, or if a stream provides the input value for another stream.

By placing of the components and adding the physical connections, a small network can be created easily. By switching to the logical view, streams can be added and configured. SlateXNS basically distinguishes between two types of streams: BE and scheduled.

After creation of a network and configuring the streams, the configuration can be validated and a schedule can be created. The validation is limited to general parameters of the network, e.g. is every host connected. Creation of a network schedule performs more detailed checks, if it is even possible to create a network schedule, e.g. are the intervals of the streams too low for the configured payload size.

The network schedule can be examined in a separate view. In this view, the gate states of each network interface of the host can be inspected. In addition, the route and the delays of the stream can be viewed. This view shows how long each packet needs to get from each talker to the corresponding listeners.

SlateXNS is also able to configure the TSN network using NETCONF. This can be done in two ways: either using the NETCONF-protocol directly or exporting the configuration as a JSON or as XML file.

User Constraints and Global Parameters

To tailor the model to the needs of the user, global parameters for the model and user constraints for the single data-streams can be configured. Inside the global parameters are optimisation options for creating the schedule, which addresses optimising End-to-End (E2E)-latency usage of priority-queues and bandwidth of BE-traffic. User constraints are more fine-grained options. The following are available:

Sending Time: defines a time period, when the transmission of a data-stream shall start

Receiving Time: defines a time period, when a data-stream shall be received

End-to-End Latency: defines the E2E-latency between start of sending a data-stream and receiving it on a certain listener

Transmission Gap: defines the gap between receiving a certain data-stream and start of sending the response, i.e. how long does it take the application to send the response.

Network Model

For demonstration purposes, a single data-stream, with an interval of 1ms was modelled. For evaluation, several sets of *Sending Time* constraints were used. No other constraints were used. In the global parameters, the optimisation of E2E-latency was selected.

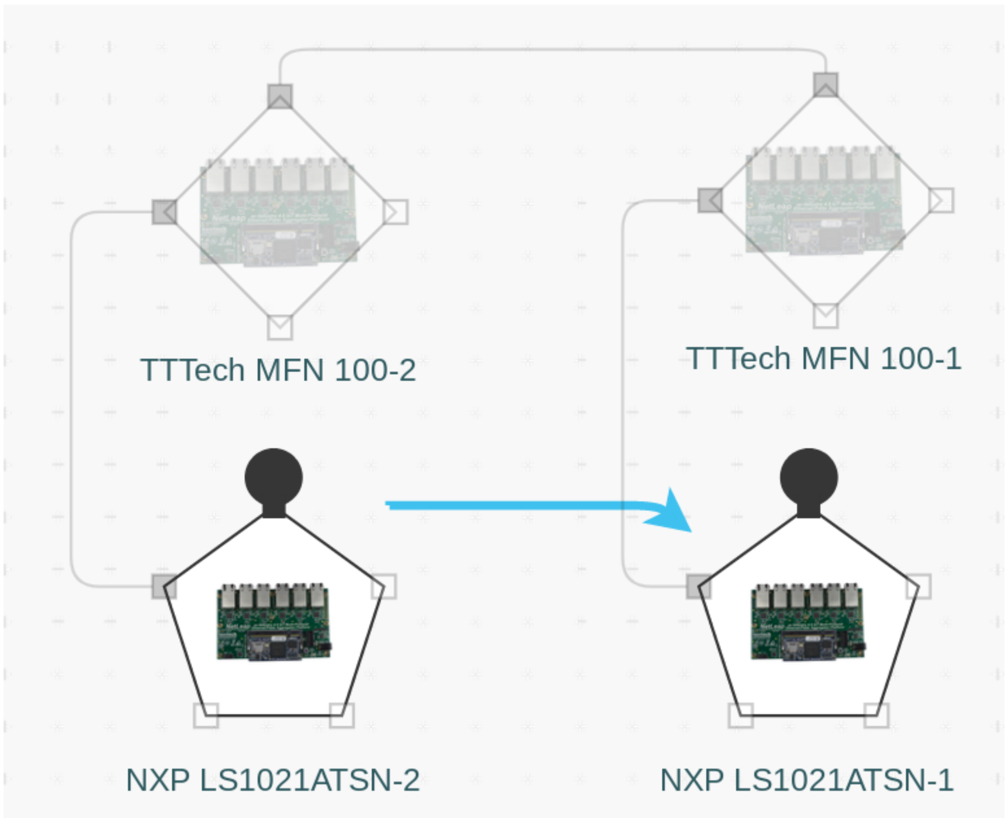

Figure 4.3: Network-Model inside SlateXNS

4.1.3 TTTech MFN100

The MFN100 is part of TTTech's Nerve platform. It consists of multiple layers. The switch, the host and a set of Virtual Machines (VMs). The block diagram of these layers is depicted in Figure 4.4. The switch is realised in an FPGA and runs a Yocto-based Operating System (OS). The host-system runs directly on the the Intel® Atom processor, here the widely used CentOS is employed. This layer also provides the resources to run the VMs for the Nerve platform.

Since only the TSN-switch is used for the project, the focus was on that layer. Configuration of the switch consists of two parts: a gate state list and the VLAN configuration. Each entry of the gate state list defines the state of a gate in a particular time interval. This state can either be open or closed. In Listing 4.1, an example gate configuration is depicted. The command *sgs*, sgs stands for *set gate state*, defines that the state of the gates shall be changed. After this, the duration of this gate state is set in nanoseconds. Due to hardware limitations, this value can't exceed 640000, i.e. 640 microseconds, on the MFN100. Next value in the line is the new state of the gates, in this case written as decimal value. The gate state is represented as an 8-bit value, where each bit is mapped to a queue, i.e. the Least-Significant Bit (LSB) is the state of queue 0 and the

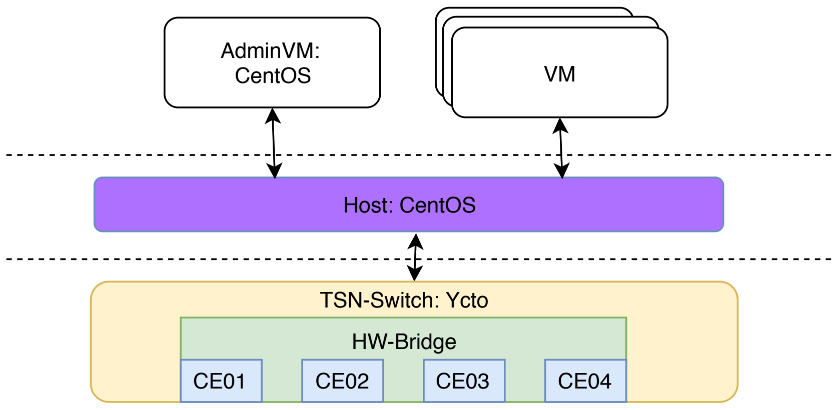

Figure 4.4: Bock diagram of MFN100

Most-Significant Bit (MSB) is the state of queue 7. If a Bit is set to *1* the gate for the queue is open, if the Bit is *0* the gate is closed. Everything after the *hash-mark* (#) is interpreted as a comment. The sum of all intervals defined in the gate state list corresponds to the cycle time, i.e. how long it takes to go through one scheduling cycle.

Listing 4.1: MFN100 example gate configuration

```
sgs 600000 3   # Timeslot: 0
sgs 20000  128 # Timeslot: 1
sgs 370000 3   # Timeslot: 2
sgs 10000  0   # Timeslot: 3
```

Configuration of MFN100

Configuration of the MFN100 is targeted to be pretty easy, especially in use with SlateXNS. Via NETCONF the MFN100 could be set up from the modelling software. However, the TSN-switch functionality of the MFN100 is still in an early development phase. The switch version available at this time did not yet support configuration via NETCONF and a workaround had to be developed. Due to the already optimised configuration format used, the conversion from NETCONF to the gate state configuration was straight forward. In Listing 4.2, a single scheduling entry of the NETCONF is depicted. This entry contains all information required for one line of the actual used configuration on the MFN100, seen in Listing 4.3.

Listing 4.2: NETCONF scheduling entry

```
<admin-control-list>
  <index>0</index>
  <operation-name>set-gate-states</operation-name>
  <sgs-params>
    <gate-states-value>3</gate-states-value>
```

```
        <time-interval-value>600000</time-interval-value>
    </sgs-params>
</admin-control-list>
```

Listing 4.3: MFN100 single gate state list entry

```
sgs 600000 3 # Timeslot: 0
```

To program the TSN-switch of the MFN100, TTTech provides a tool called *tsntool*. The first command in Listing 4.4 uses the prior created gate state list and applies it to an interface, in this case interface *CE03*. To complete the configuration, additional timing information has to be added to the interface. This is done by the second command in Listing 4.4. This command consists of four parts: the *base time*, the *cycle time*, the *cycletime-extension* and the interface the data shall be applied to. The base time defines when the schedule shall be started, it is given as seconds.nanoseconds. This parameter can be provided either as an absolute value, defining a certain point in time, or as a relative value, defining the offset to the current time. If the value shall be interpreted as a relative value, a leading + or − is required. The cycle time defines how long one cycle of the schedule takes. It is given as a fraction of seconds. The cycle time extension defines how long the current gating cycle is extended when updating the configuration. This is given in nanoseconds. The final parameter is the interface to which the configuration shall be applied to.

Listing 4.4: tsntool configuration

```
tsntool st wrcl MFN-100-1_CE03.cfg CE03
tsntool st configure 0.0 1/100 100 CE03
```

To enable routing of VLANs the Linux-system has to be configured accordingly. The command in Listing 4.5 was used to configure the VLAN 100 for the interface CE03.

Listing 4.5: MFN100 VLAN configuration

```
bridge vlan add dev CE03 vid 100
```

4.1.4 NXP LS1021ATSN

In contrast to the MFN100, NXP LS1021ATSN is a development board and not designed to be used in a productive environment. The heart of this board is a LS1021A SoC with an included SJA1105TEL TSN-switch.

Since the NXP LS1021ATSN is not only a TSN end system, but also contains a switch, both systems need to be configured. To ease the understanding of the configuration, the block diagram of the LS1021ATSN is depicted in Figure 4.5. In addition to the original naming of the individual network interfaces on the chassis, the names displayed in Linux were annotated. The information of the mapping of the switch-ports is depicted in Table 4.1. This mapping is also relevant for SlateXNS and this needs to be respected when creating a separated component for the LS1021ATSN.

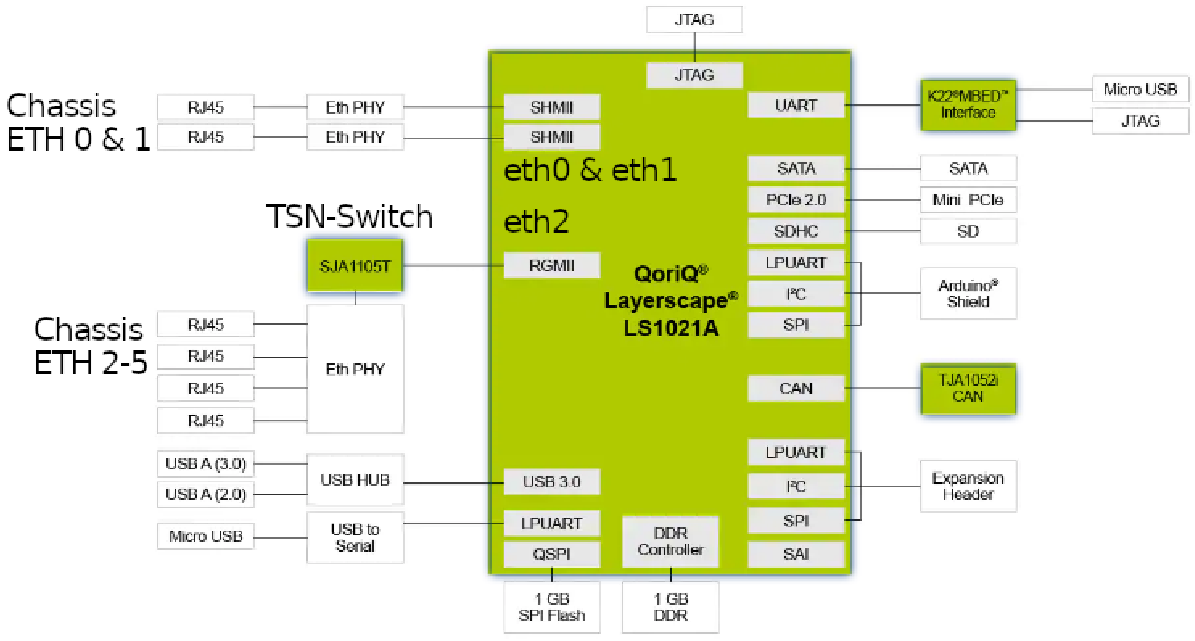

Figure 4.5: Block Diagram of the LS1021A SoC

Port location	Switch Port
Chassis ETH2	1
Chassis ETH3	2
Chassis ETH4	3
Chassis ETH5	0
To LS1021	4

Table 4.1: Mapping of SJA1105TEL switch ports to the actual board

Configuration of NXP SJA1105 TSN-switch

NXP provides a set of command-line tools for creating configurations and setting up the TSN-switch. These tools are open-source and available on Github[1].

From the set of tools, only the *sja1105-tool* and the *schedule-create* scripts were used. The *sja1105-tool* is a program which is used to modify the current configuration and transmitting it to the switch. For the purpose of creating or modifying configurations, the tool holds the configuration, which is currently worked on, internally. This tool can be run on any PC with a Linux-based OS. The result can be exported as an XML-file, which can either be loaded directly on the hardware or transmitted via NETCONF. The *schedule-create* script, is a wrapper which reads a schedule written in JSON-format. An example for the JSON gate-configuration can be found in Listing 4.6. Beginning

[1] `https://github.com/openil/sja1105-tool` accessed 17.08.2019

at the outermost layer, there are two keys, *clksrc* and *cycles*. *clksrc* stands for clock source, thus, defining which clock is used. In this case, only the local clock, without any synchronisation is used. For synchronisation, either *none*, *standalone*, *PTP* or the clock synchronisation algorithm defined by SAE AS6802 [AS616] (TTEthernet) can be used. The *cycles* array is used to configure the schedule, each entry in this array corresponds to one sub-schedule. The switch can handle up to eight sub-schedules. The *start-time-ms*, defines the delay when this sub-schedule is started. This value has to be a multiple of 200 nanoseconds and is not allowed to overlap with any other sub-schedule. Now, the key *timeslots* configures the gates of each port. Each timeslot has a *duration-ms* key, which again has to be a multiple of 200 nanoseconds and defines how long the timeslot is active. The *ports* key is used to define the affected ports by this entry. In this example, it is defined that all ports are always affected. Additional information about the mapping of the switch-ports of the labels on the board and to the LS1021A SoC is provided in Table 4.1. The next key, *gates-open*, sets which transmission gates of the defined ports are open during this cycle. The last key, *comment*, is optional and can be used for documentation purposes, the tool ignores this entry.

Listing 4.6: SJA1105TEL example schedule

```
{
  "clksrc": "ptp",
  "cycles": [
    {
      "start-time-ms": "1",
      "timeslots": [
        {
          "duration-ms": "20",
          "ports": [0,1,2,3,4],
          "gates-open": [0,1,2,3,4,5,6],
          "comment": "BE Traffic"
        },
        {
          "duration-ms": "10",
          "ports": [0,1,2,3,4],
          "gates-open": [7],
          "comment": "Priority Traffic"
        },
        {
          "duration-ms": "1",
          "ports": [0,1,2,3,4],
          "gates-open": [],
          "comment": "Guard Band"
        }
      ]
    }
    ]
}
```

A full list of configurable options of the switch can be found in a user manual provided by NXP[2].

For easier use of the prior defined tooling, the script in Listing 4.7 was created. This script configures the VLANs, mirroring ports and the schedule of the TSN-switch. In Line 2, the currently loaded configuration is discarded and the default is loaded. This is done in order to have a clean state for the start of the configuration. The for-loop, lines 5 to 12, is used to set the allowed traffic to the maximum and traffic mirroring is activated on each port. The lines 14 to 19 handle the configuration of the used VLAN, in this case VLAN 100. The command in line 22 sets the output of the mirrored traffic to interface *3*, which is marked on the chassis as *ETH4*. The command in line 24 calls the *schedule-create* script to read the JSON configuration and feed the data to the *sja1105-tool*. In line 27, the created configuration is exported as to be transmitted to the target.

Listing 4.7: NXP SJA1105 configuration script

```
1  ...
2  sja1105-tool config default ls1021atsn
3
4  # Extend ingress policer MTU to include VLAN tag
5  for port in $(seq 0 4); do
6          for prio in $(seq 0 7); do
7                  policer-limit --port ${port} --prio ${prio} \
8                  --mtu 1522 --rate-mbps 1000
9          done
10         sja1105-tool config modify mac-config[${port}] egr_mirr 1
11         sja1105-tool config modify mac-config[${port}] ing_mirr 1
12 done
13
14 sja1105-tool config modify vlan-lookup-table entry-count 2
15 # Create VLAN ID 100 on switch
16 sja1105-tool config modify vlan-lookup-table[1] vmemb_port 0x1F
17 sja1105-tool config modify vlan-lookup-table[1] vlan_bc    0x1F
18 sja1105-tool config modify vlan-lookup-table[1] vlanid     100
19 sja1105-tool config modify vlan-lookup-table[1] tag_port   0x1F
20
21 # set mirror output
22 sja1105-tool config modify general-parameters-table mirr_port 3
23
24 scheduler-create --file ${board_config}
25
```

[2] https://www.nxp.com/docs/en/user-guide/UM10944.pdf accessed 17.08.2019

```
rm -f ${output}
sja1105-tool config save ${output}
```

4.1.5 Configuration of Open Industrial Linux

The general set up of OpenIL on the board is out of scope of the thesis, the corresponding information can be found in the user manual[3].

Since there is currently no TSN network stack publicly available, other OS interfaces have to be used and reconfigured to work with TSN. First, a VLAN has to be set up, so that the sent frames get tagged for the switches. In general there are no limitations for the VLAN Identification (VID), except the VIDs zero and 4095.

Second, a mapping from network queue priority to VLAN has to be added. How the mapping is done, is not relevant, in this case direct mapping is used. Network queue priority one equals to VLAN priority one and so on. In Listing 4.8 VLAN 100 is configured at interface *eth*2. As name for the new interface *eth*2.100 was chosen.

Listing 4.8: LS1021ATSN example VLAN configuration

```
ip link add link eth2 name eth2.100 type vlan id 100 egress \
0:0 1:1 2:2 3:3 4:4 5:5 6:6 7:7
```

Linux sockets are a very low level interface to the network stack. This allows access to options typically not available on higher level Application Programming Interfaces (APIs). Via the *setsockopt* function, the socket buffer priority can be adapted, which is used as priority in the network stack. Listing 4.9 shows how to set the priority of socket *socket_fd* to the highest value. Additional information about the specific *setsockopt* arguments can be found in the corresponding Linux manual pages.

Listing 4.9: Linux C-socket configuration

```
int optval=7 // defines the priority which shall be given
             // min. value 1, max. value 7
             // higher value => higher priority
setsockopt(socket_fd, SOL_SOCKET, SO_PRIORITY, &optval, \
           sizeof(optval))
```

4.1.6 Management Network

The management network, which is also illustrated in Figure 4.2, is used for configuration and network analysis.

In Listing 4.10 a command is depicted to debug network traffic on remote interfaces. In this example the *eth2* interface of OpenIL is captured. Via tcpdump the capturing of

[3] `https://www.nxp.com/docs/en/user-guide/OPEN-LINUX-IND-UM-1-5.pdf` accessed 31.08.2019

the in- and outgoing traffic is done on the target. The output of the tool was forwarded to be displayed inside Wireshark on the management PC.

Listing 4.10: Remote capturing using tcpdump and wireshark

```
ssh root@192.168.8.101 "/usr/sbin/tcpdump␣-i␣eth2␣-U␣-w␣-␣\
'not␣(host␣192.168.8.101␣and␣port␣22)'" | wireshark -i - -k
```

4.2 Evaluation

For demonstration, time synchronisation and network delays in TSN-based networks are shown.

4.2.1 Synchronising the Network

All devices of the demonstrator provide the *linuxptp*-stack for synchronisation, which provides three tools:

ptp4l: is used to set up the PTP-stack and start the time synchronisation of the network hardware.

phc2sys: is used to synchronise the system clock to the network.

pmc: is a management client to extract additional information from the PTP-stack.

To be conform to the gPTP specification, Peer-to-Peer (P2P)-delay measurement on OSI-model layer 2 was configured. For the two MFN100, PTP was executed on interfaces *CE03* and *CE04* and on the LS1021ATSN boards only on interface *eth2*. The complete configuration can be found in Appendix A.1. In Listing 4.11, the usage of *ptp4l* is depicted. The $-f$ defines the configuration file to be used, $-m$ defines to print the output on the console and $-l$ 7 sets the log level to the maximum value. The parameter $-t$ is an extension from OpenIL, it defines the maximum offset of the local clock, in this case to 200 ns.

Listing 4.11: LS1021ATSN example ptp4l configuration

```
ptp4l -f ptp4l_ls1021atsn.cfg -m -l 7 -t 200
```

To synchronise the system clock to the network, *phc2sys* was used, which can be read as *physical clock to system clock*. The used settings are depicted in Listing 4.12. The usage is quite similar to *ptp4l*, $-m$ and $-l$ have the same meaning. The parameter $-a$ directs the tool to do the synchronisation automatically. $-r$ instructs to synchronise the system clock, typically *CLOCK_REALTIME* in Linux-based systems. The repeated use of $-r$ defines that the system clock is also eligible to be a time source.

Listing 4.12: LS1021ATSN example phc2sys configuration

```
phc2sys -a -rr -m -l 7
```

To check if all clocks are synchronised, *pmc* can be used to read out data from the PTP-stack. In Listing 4.14, the output of the PTP grandmaster is depicted and Listing 4.15 shows the output of a device which is synchronised to the grandmaster. The devices are synchronised if all devices have the same value as *gmIdentity.* The grandmaster is the only device where the *gmPresent* flag is set to *false.*

Listing 4.13: Check PTP time status

```
pmc -u -b 0 'GET TIME_STATUS_NP'
```

Listing 4.14: PTP grandmaster time output

```
sending: GET TIME_STATUS_NP
  8823fe.fffe.012ed4-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
    master_offset              0
    ingress_time               0
    cumulativeScaledRateOffset +0.000000000
    scaledLastGmPhaseChange    0
    gmTimeBaseIndicator        0
    lastGmPhaseChange          0x0000'0000000000000000.0000
    gmPresent                  false
    gmIdentity                 8823fe.fffe.012ed4
```

Listing 4.15: PTP slave time output

```
sending: GET TIME_STATUS_NP
  00049f.fffe.ef0602-0 seq 0 RESPONSE MANAGEMENT TIME_STATUS_NP
    master_offset              -7
    ingress_time               1566398835877789937
    cumulativeScaledRateOffset +0.000000017
    scaledLastGmPhaseChange    0
    gmTimeBaseIndicator        0
    lastGmPhaseChange          0x0000'0000000000000000.0000
    gmPresent                  true
    gmIdentity                 8823fe.fffe.012ed4
```

In Figure 4.6, a measurement of the time synchronisation of the demonstration network is shown. In the top-most diagram the local clock offset in nanoseconds of each clock is printed. When evaluating the offset over a set of samples, the quality of clocks can be compared. It is worth noting that the MFN100 devices perform considerably better than the LS1021A boards.

The remaining diagrams in Figure 4.6 depict the adjustment of the frequency from the local clock in parts per million (ppm). The frequency is changed in order to reduce the jitter of local clocks and to keep the offset low. Both of the MFN100 switches start with a larger frequency offset than the LS1021A boards. Therefore, they reduce the clock speed. The LS1021A boards have a slower ticking clock and do not need to adapt their frequency as much. Due to the lower quality of the clock, there is more jitter here. In the graph of "MFN100-1" a spike down is visible. This outlier is a test point where the internal clock had a so-called *jump*. When a clock jumps, the timestamp is outside certain bounds and a resynchronisation has to be done.

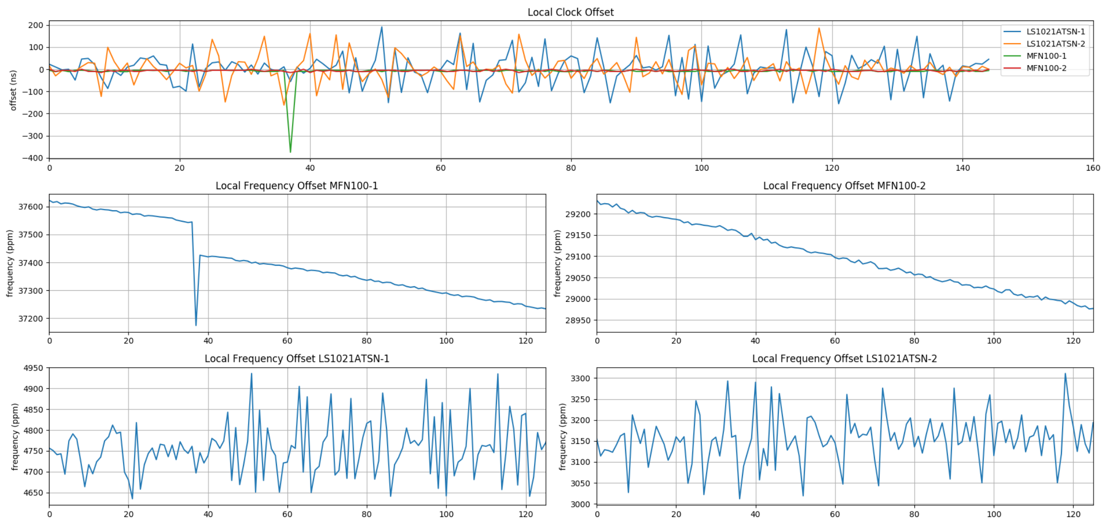

Figure 4.6: Synchronisation of clocks via PTP

4.2.2 Qbv synchronised Schedule

After finishing synchronisation of all network devices, the TSN schedule has to be applied.

Testing Method

For measuring the time a package takes from one endpoint to the next one, a C++ program was developed. C++ was selected because of the easy access to the socket-API of Linux and the typically low overhead added by the programming language. For precise measurements, Linux provides a time API.

A simple ping program was implemented to measure the delay between both endpoints based on User Datagram Protocol (UDP). The message consists of timestamp taken from *CLOCK_REALTIME*, since this clock is synchronised to the network. On the receiver side the received timestamp is compared to the time of reception and the delta in milliseconds is calculated.

To show the benefit of TSN, a high network load is applied, for this *iperf3* was employed. iperf3 is a network bandwidth test tool and can output a sustained network load over a given period of time. Since there were some hardware issues when the Gbit network was put to a stress test, the decision was made to limit the network to 100 Megabits per second(Mbit/s) bandwidth. This was done by configuring the connection between the MFN100 switches to a lower speed, see Listing 4.16

Listing 4.16: Limiting network speed of interface

```
ethtool -s CE04 speed 100 duplex full autoneg off
```

Results

In Figure 4.7, a network delay comparison between TSN-traffic and BE-traffic can be found. In the top diagram the network delay of the last 1000 samples is printed. Clearly visible in this line graph is the lower network delay and less jitter in the TSN-traffic. Rarely, a TSN message misses its intended time-slot as the sending application is executed on a non-real-time OS. The message is then transmitted in the next time-slot, which causes an additional delay of 1 ms. The BE-traffic has a much higher delay and higher jitter. The bottom diagram is a histogram over 11000 samples of the end-to-end network delay. It shows more clearly the expected low jitter of TSN traffic, indicated by a well-shaped spike at around 0.5 ms. In comparison the BE-traffic has a network delay distribution from 1.5 to over 4 ms.

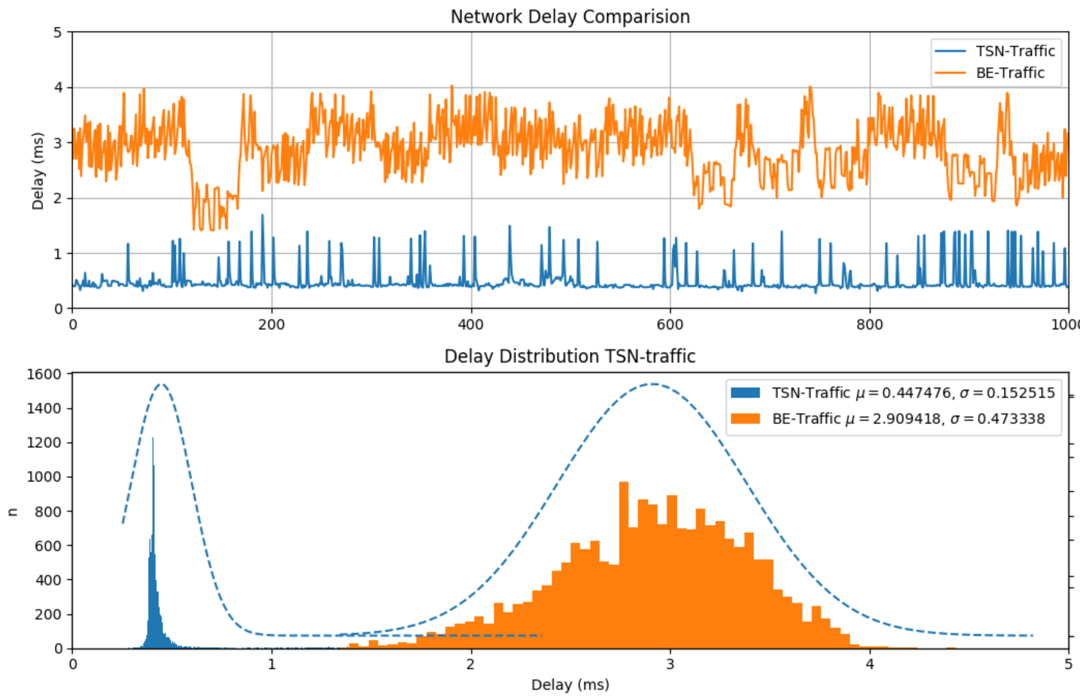

Figure 4.7: Network delay comparison between TSN and BE-traffic

4.3 Limitations and Pitfalls

When working with the currently available TSN-solutions some pitfalls have to be kept in mind.

4.3.1 SlateXNS and TTTech MFN100

At the time of writing, the configuration of the MFN100 via NETCONF is not possible. Through additional support material from TTTech, this could be bypassed. Additionally, an upper limit of 640 us per timeslot has to kept in mind.

4.3.2 NXP LS1021ATSN

Since there was no accurate model for the LS1021ATSN board available, a separate device configuration had to be created. The switch has some additional properties, which are not clearly defined in its user manual. The important one is that scheduling events are not allowed to occur simultaneously. Therefore, between every event there has to be at least one tick, which corresponds to 200 nanoseconds. In the demonstration network, this condition can be tolerated, since an intermediate timeslot could be inserted without affecting the set up.

The automatic negotiation of the network speed does not work, in this case, the SJA1105 TSN-switch is configured by default to use Gbit-Ethernet. If network devices which do not provide Gbit-Ethernet are used, the ports need to be reconfigured. OpenIL provides a script to mitigate this shortcoming[4].

4.3.3 Network Stack Delays

Currently, SlateXNS is not aware of the delay of the network stack used. This has to be kept in mind when running the software. In Figure 4.8, five data-streams are compared to each other. All streams are transmitted using a TSN schedule with 1 ms cycle time. Messages are handed over to the Linux network stack at the start of each cycle. However, each of the five streams has a different transmission window associated, opening at 100 - 900 us after the cycle start. Clearly visible is that the stream with a transmission window opening at 100 us always misses its dedicated slot and uses the next one, which results in a higher network delay. The stream, with a transmission window opening at 300 us, is evenly split between two slots in the diagram. This indicates that missing this slot has a probability of about 50%. This analysis shows that the Linux network stack delay is an important factor, when designing TSN applications.

4.3.4 Network Stack configuration

Without a TSN network stack which obeys the schedule at the endpoints, the applications have to take care of this. Since one of the key points of TSN is keeping the jitter for

[4] `https://github.com/openil/sja1105-tool#known-issues` accessed 31.08.2019

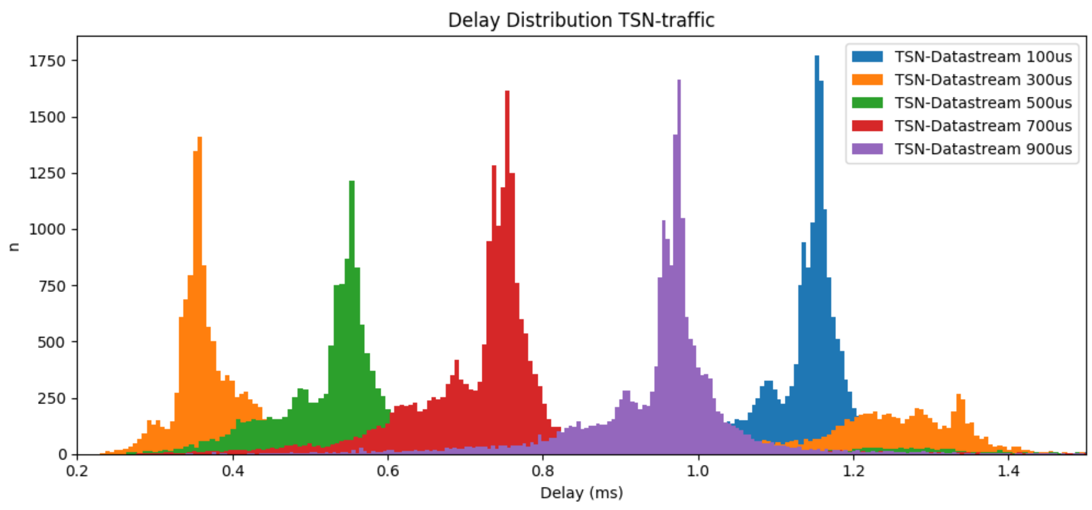

Figure 4.8: Network delay comparison between five data-streams

messages low, just sending the frames and letting the switches do their part may be insufficient for certain applications.

In Figure 4.9, their are two independent streams using queue seven. Without any additional identification mechanism, the network stack is not aware which socket corresponds to which time slot/queue. This can lead to unexpected high jitter.

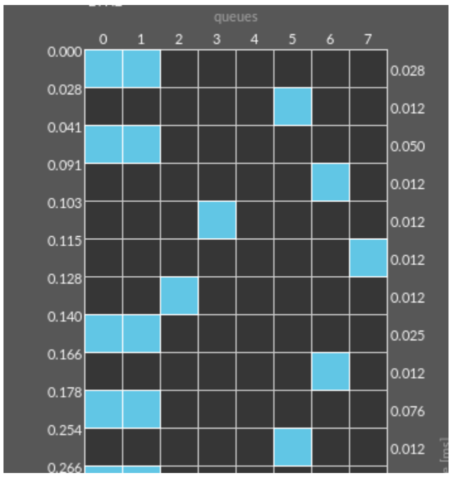

Figure 4.9: Not identifiable stream SlateXNS

In case of the NXP SJA1105 switch, Virtual Links (VLs) could be used, but these are not part of TSN. They have their origin in TTEthernet. In IEEE 802.1Q and 802.1CB, a so-called StreamID is defined, to uniquely identify the frames of a stream. This identifiers have to be known in advance and the network stack has to add/interpret them.

In order to configure the network stacks, more advanced modelling tools are required. A software developer just wants to send a frame and the middleware or the communication stack should take care of actually sending the frame. As seen in Figure 4.9, multiple streams use the same queue, but no StreamID is configured by SlateXNS. At the end points, there has to be a stack which knows which stream has to be sent at which point in time.

CHAPTER 5

Conclusion

In this thesis, several standards contributing to Time-Sensitive Networking (TSN) have been evaluated. To show the current state of this new technology a demonstration network was built. With the tool SlateXNS, the network was modelled and schedules were created. Precision Time Protocol (PTP) was used to synchronise the single devices to each other. This demonstration network was then used to conduct measurements on the claims of TSN.

With the newly introduced timeslots and queues for Ethernet-based networks, the traversal of messages through a network was improved. As shown in Section 4.2.2, the jitter and the network delay could be reduced drastically by TSN. This shows the potential of TSN.

The used demonstration network had quite little capabilities. In future work, the evaluation of modelling of larger networks will be an interesting topic. An additional topic here is to have a reliable method for hardware configuration. Both suppliers of the used hardware used different models. In the future, when the methods defined in the standards have been implemented, configuring these networks is expected to be easier and have less pitfalls. For reliable data transportation, Frame Replication and Elimination for Reliability (FRER) is a big topic. By expanding the demonstration network with future generations of the used hardware, FRER can be put under test. Another interesting topic is transferring the gathered knowledge from the demonstrator in real world application like industrial automation and automotive networks.

APPENDIX A

Configuration Files

A.1 ptp4l configuration files

Listing A.1: MFN100 ptp4l config

```
[global]
#
# Default Data Set
#
twoStepFlag                1
gmCapable                  1
slaveOnly                  0
priority1                  246
priority2                  246
domainNumber               0
clockClass                 187
clockAccuracy              0x23
offsetScaledLogVariance    0xFFFF
free_running               0
freq_est_interval          1
dscp_event                 0
dscp_general               0
#
# Port Data Set
#
logAnnounceInterval        0
logSyncInterval           -3
logMinDelayReqInterval    -1
logMinPdelayReqInterval   -1
```

```
announceReceiptTimeout        3
syncReceiptTimeout            3
delayAsymmetry                0
fault_reset_interval          4
neighborPropDelayThresh       20000000
min_neighbor_prop_delay      −20000000
#
# Run time options
#
assume_two_step               1
logging_level                 6
path_trace_enabled            1
follow_up_info                1
hybrid_e2e                    0
tx_timestamp_timeout          2
use_syslog                    1
verbose                       1
summary_interval             −3
kernel_leap                   1
check_fup_sync                0
#
# Servo Options
#
pi_proportional_const         0.0
pi_integral_const             0.0
pi_proportional_scale         0.0
pi_proportional_exponent     −0.3
pi_proportional_norm_max      0.7
pi_integral_scale             0.0
pi_integral_exponent          0.4
pi_integral_norm_max          0.3
step_threshold                0.0
first_step_threshold          0.00002
max_frequency                 900000000
clock_servo                   pi
sanity_freq_limit             200000000
ntpshm_segment                0
#
# Transport options
#
transportSpecific             0x0
ptp_dst_mac                   01:80:C2:00:00:0E
p2p_dst_mac                   01:80:C2:00:00:0E
```

```
udp_ttl                    1
udp6_scope                 0x0E
uds_address                /var/run/ptp4l
#
# Default interface options
#
network_transport          L2
delay_mechanism            P2P
time_stamping              hardware
tsproc_mode                filter
delay_filter               moving_median
delay_filter_length        10
egressLatency              0
ingressLatency             0
boundary_clock_jbod        1
#
# Clock description
#
productDescription         ;;
revisionData               ;;
manufacturerIdentity       00:00:00
userDescription            ;
timeSource                 0xA0

[CE03]
[CE04]
```

Listing A.2: LS1021ATSN ptp4l config

```
[global]
#
# Default Data Set
#
twoStepFlag              1
gmCapable                1
slaveOnly                0
priority1                248
priority2                248
domainNumber             0
clockClass               248
clockAccuracy            0xFE
offsetScaledLogVariance  0xFFFF
free_running             0
freq_est_interval        1
```

```
dscp_event                    0
dscp_general                  0
#
# Port Data Set
#
logAnnounceInterval           0
logSyncInterval              −3
logMinDelayReqInterval       −1
logMinPdelayReqInterval      −1
announceReceiptTimeout        3
syncReceiptTimeout            3
delayAsymmetry                0
fault_reset_interval          4
neighborPropDelayThresh       800
min_neighbor_prop_delay      −20000000
#
# Run time options
#
assume_two_step               1
logging_level                 6
path_trace_enabled            1
follow_up_info                1
hybrid_e2e                    0
tx_timestamp_timeout          2
use_syslog                    1
verbose                       1
summary_interval             −3
kernel_leap                   1
check_fup_sync                0
#
# Servo Options
#
pi_proportional_const         0.0
pi_integral_const             0.0
pi_proportional_scale         0.0
pi_proportional_exponent     −0.3
pi_proportional_norm_max      0.7
pi_integral_scale             0.0
pi_integral_exponent          0.4
pi_integral_norm_max          0.3
step_threshold                0.0
first_step_threshold          0.00002
max_frequency                 900000000
```

```
clock_servo                 pi
sanity_freq_limit           200000000
ntpshm_segment              0
#
# Transport options
#
transportSpecific           0x0
ptp_dst_mac                 01:80:C2:00:00:0E
p2p_dst_mac                 01:80:C2:00:00:0E
udp_ttl                     1
udp6_scope                  0x0E
uds_address                 /var/run/ptp4l
#
# Default interface options
#
network_transport           L2
delay_mechanism             P2P
time_stamping               hardware
tsproc_mode                 filter
delay_filter                moving_median
delay_filter_length         10
egressLatency               0
ingressLatency              0
boundary_clock_jbod         1
#
# Clock description
#
productDescription          ;;
revisionData                ;;
manufacturerIdentity        00:00:00
userDescription             ;
timeSource                  0xA0

[eth2]
```

List of Figures

List of Tables

Listings

Glossary

ARM® ARM is a processor architecture mostly used for mobile devices and embedded devices. ARM stands for "Advanced RISC Machines". 17, 18

Avnu Alliance The Avnu Alliance is a community creating an interoperable ecosystem servicing the precise timing and low latency requirements of diverse applications using open standards through certification. `https://avnu.org/`. 3

bridge A system that includes Media Access Control (MAC) Bridge or Virtual Local Area Network (VLAN) Bridge component functionality. [80218a]. 5, 7, 9–12, 14

C is a general purpose programming language, often used embedded systems or programming of Operating Systems. 32, 53

C++ is an extension to C, which provides support for classes and other object-oriented programming techniques. 35

end station A device attached to a local area network (LAN) or metropolitan area network (MAN), which acts as a source of and/or destination for data traffic carried on the LAN or MAN. (From IEEE Std 802) [80217c]. 10

end system A system attached to a network that is an initial source or a final destination of packets transmitted across that network. [80217c]. 10, 17, 20, 21, 24

Ethernet Is a protocol which operates on the layer 2 of the OSI-model. It is defined standard IEEE 802.3 . 1, 3, 5, 13–20, 37, 41

Fronthaul The connectivity between the functional blocks (e.g., baseband processing and radio frequency blocks) of a cellular base station. [80218b]. 15, 18

grandmaster The time-aware system that contains the best clock, as determined by the best master clock algorithm (BMCA), in the generalized precision time protocol (gPTP) domain. [80211]. 4, 7, 8, 34, 53

JSON JavaScript Object Notation, a human-readable file format which consists of key-value pairs. 25, 29, 31

layer 3-switch In comparission with a typical switch, this device adds some kind of routing capabilities, for instance the handling of VLANs. 11

Linux Linux is the superset of open-source operating systems based on the Linux kernel. 18, 28, 29, 32, 33, 35, 37, 53

listener The end station that is the destination, receiver, or consumer of a stream. [80218a]. 3, 9, 10, 25

member stream A stream that is linked with other member streams via Frame Replication and Elimination for Reliability (FRER) to form a Compound Stream. [80217c]. 10, 11

. 5, 11, 33, 55

ProfiBus Process Field Bus, is a field bus developed, by the German department of education and research, for industrial automation. 1

router A router is a device which is located on OSI-Layer 3. Routers handle the routing of packets between networks. 5, 11

SSH Secure Shell provides a protection layer to access services securly on unsecure networks. Typically used for remote command execution or login into devices. In general, SSH can be added to any network service. 6

stream A unidirectional flow of data (e.g., audio and/or video) from a talker to one or more Listeners. [80218a]. 3, 9, 10, 12, 13, 24, 25, 37, 38, 49

switch A switch has multiple network ports, where each port provides bridge functionality. xiv, 5, 10, 17–20, 23, 24, 26–32, 35–38, 51

talker The end station that is the source or producer of a stream. [80218a]. 3, 9, 10, 25

tcpdump is a commandline tool for packet analysis and troubleshooting. 32

time-aware bridge A Bridge that is capable of communicating synchronized time received on one port to other ports, using the IEEE 802.1AS protocol. [80211]. 3, 7

time-aware end station An end station that is capable of acting as the source of synchronized time on the network, or destination of synchronized time using the IEEE 802.1AS protocol, or both. [80211]. 3, 7

time-aware system A time-aware Bridge or a time-aware end station. [80211]. 7, 8

traffic class A classification used to expedite transmission of frames generated by critical or time-sensitive services. Traffic classes are numbered from zero through N-1, where N is the number of outbound queues associated with a given Bridge port, and 1 <= N <= 8, and each traffic class has a one-to-one correspondence with a specific outbound queue for that port. Traffic class 0 corresponds to nonexpedited traffic; nonzero traffic classes correspond to expedited classes of traffic. A fixed mapping determines, for a given priority associated with a frame and a given number of traffic classes, what traffic class will be assigned to the frame. [80218a]. 5, 13

transmission gate A gate that connects or disconnects the transmission selection function of the forwarding process from the queue, allowing or preventing it from selecting frames from that queue. The gate has two states, Open and Closed. [80218a]. 13

VLAN The closure of a set of MAC Service Access Points (MSAPs) such that a data request in one MSAP in the set is expected to result in a data indication in another MSAP in the set. [80218a]. 12, 24, 26, 28, 31, 32, 62

Wireshark is a packet analyzer, which is often used for troubleshooting and network analysis. 33

XML Extensible Markup Language, a markup-language which defines rules for encoding to be human and machine-readable. 6, 25, 29

YANG YANG (Yet another next Generation) is a modelling language, in combination with NETCONF it is used for modelling configurations.. 6

Yocto An open-source projects, which goal is to provide custom-built Linux-Images for embedded projects. 26

Acronyms

AD Autonomous Driving. 1, 15

ADAS Advanced Driver Assistance Systems. 1, 15

API Application Programming Interface. 32, 35

ASIL Automotive Safety Integrity Level. 19, 21

AV Audio-Video. 14

AVB Audio Video Bridging. 14–21

BBU Baseband Unit. 15

BE Best Effort. ix, xi, 4, 5, 25, 36, 49

BMCA Best Master Clock Algorithm. 7, 8

CAN Controller Area Network. 1, 4, 5

CNC Centralized Network Configuration. 10

CPRI Common Public Radio Interface. 15

CPU Central Processing Unit. 18

CQF Cyclic queuing and forwarding. 13

CSMA Carrier-Sense Multiple Access. 4

CSMA-CD Carrier-Sense Multiple Access - Colission Detection. 5

CSMA-CR Carrier-Sense Multiple Access - Colission Resolution. 4, 5

CUC Centralized User Configuration. 10

DHCP Dynamic Host Configuration Protocol. 5

DNS Domain Name System. 5

DPI Deep Packet Inspection. 18

E2E End-to-End. 25

ECU Electronic Control Unit. 17, 19

FIT Faults in Time. 19

FPGA Field-Programmable Gate Array. 19, 26

FRER Frame Replication and Elimination for Reliability. 10, 12, 41

Gbit Gigabit. 18, 20, 21, 35, 37

Gbit/s Gigabit per second. 1, 18, 20, 21

GPS Global Positioning System. 4

gPTP generalized Precision Time Protocol. 3, 6, 7, 13, 14, 33

IC Integrated Circuit. 18

IEC International Electrotechnical Commission. 15

IEEE Institute of Electrical and Electronics Engineers. 1, 3, 6, 8–10, 12, 13, 15, 38, 51

IIC Industrial Internet Consortium. 3

IIoT Industrial Internet of Things. 1

IoT Internet of Things. 1

IP Internet Protocol. 5

IPV Internal Priority Value. 13

IS-IS Intermediate System to Intermediate System. 11

IT Information Technology. 1

LAN Local Area Network. 7

LSB Least-Significant Bit. 26

MAC Media Access Control. 4, 11, 12, 57

Mbit/s Megabit per second. 36

MCU Microcontroller Unit. 17–19

MIB Management Information Base. 6

MSB Most-Significant Bit. 27

MSRP Multiple Stream Registration Protocol. 10, 11

NETCONF Network Configuration. 6, 24, 25, 27, 29, 37

NTP Network Time Protocol. 4

OpenIL Open Industrial Linux. 18, 32, 33, 37

OS Operating System. 26, 29, 32, 36

OSPF Open Shortest Path First. 11

OT Operational Technology. 1, 15

P2P Peer-to-Peer. 33

PC Personal Computer. 20, 29

PCE Path Computation Element. 11

PCIe Peripheral Component Interconnect Express. 15, 19, 20

PCR Path Control and Reservation. 11

PLC Programmable Logic Controller. 17, 19

ppm parts per million. 34

PSFP Per-Stream Filtering and Policing. 12, 13

PTP Precision Time Protocol. 4, 7, 9, 30, 33–35, 41, 49

RAN Radio Access Network. 15

RPC Remote Procedure Call. 6

RRH Remote Radio Head. 15

SNMP Simple Network Management Protocol. 5, 6

SoC System on a Chip. 17–19, 28–30, 49

SPB Shortest Path Bridging. 11

SRP Stream Reservation Protocol. 3, 9

STP Spanning Tree Protocol. 11

TDMA Time-Division Multiple Access. 5, 13

TSN Time-Sensitive Networking. ix, xi, xiv, 1–3, 5, 6, 8–10, 13–21, 23–29, 31–33, 35–38, 41, 49, 51

TTEthernet Time-Triggered Ethernet. 5, 16, 18, 30, 38

UDP User Datagram Protocol. 35

UNCI User/Network Configuration Information. 9, 10

UNI User/Network Interface. 3, 9

VID VLAN Identification. 32

VL Virtual Link. 38

VM Virtual Machine. 26

Bibliography

[80211] IEEE standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks. *IEEE Std 802.1AS-2011*, pages 1–292, March 2011.

[80216a] IEEE standard for local and metropolitan area networks – bridges and bridged networks – amendment 26: Frame preemption. *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pages 1–52, Aug 2016.

[80216b] IEEE standard for local and metropolitan area networks— bridges and bridged networks - amendment 24: Path control and reservation. *IEEE Std 802.1Qca-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qcd-2015 and IEEE Std 802.1Q-2014/Cor 1-2015)*, pages 1–120, March 2016.

[80216c] IEEE standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic. *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q— as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q—/Cor 1-2015)*, pages 1–57, March 2016.

[80217a] IEEE standard for local and metropolitan area networks–bridges and bridged networks–amendment 28: Per-stream filtering and policing. *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pages 1–65, Sep. 2017.

[80217b] IEEE standard for local and metropolitan area networks–bridges and bridged networks–amendment 29: Cyclic queuing and forwarding. *IEEE 802.1Qch-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd(TM)-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, IEEE Std 802.1Qbz-2016, and IEEE Std 802.1Qci-2017)*, pages 1–30, June 2017.

[80217c] IEEE standard for local and metropolitan area networks–frame replication and elimination for reliability. *IEEE Std 802.1CB-2017*, pages 1–102, Oct 2017.

[80218a] IEEE standard for local and metropolitan area network–bridges and bridged networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pages 1–1993, July 2018.

[80218b] IEEE standard for local and metropolitan area networks – time-sensitive networking for fronthaul. *IEEE Std 802.1CM-2018*, pages 1–62, June 2018.

[80218c] IEEE standard for local and metropolitan area networks–bridges and bridged networks – amendment 31: Stream reservation protocol (srp) enhancements and performance improvements. *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pages 1–208, Oct 2018.

[AS616] *SAE AS6802 - Time-Triggered Ethernet*, nov 2016.

[MEF13] The MEFForum. Mef 10.3 technical specification Ethernet services attributes phase 3. In *MEF 10.3 Technical Specification Ethernet Services Attributes Phase 3*, 2013.