Towards Dynamic Time-Sensitive Networking (TSN) for the Factories of the Future

- Communication Technologies -

Project Worksheets

Group 750

Aalborg University
Department of Electronic Systems
Fredrik Bajers Vej 7B
DK-9220 Aalborg

Preface

This is a collection of worksheets made for a 7th semester project in Communication Technology on Aalborg University.

These worksheets should not necessarily be read chronologically, but does follow the structure of the paper. It is however recommended to read chapter 1 *802.1Q Time-Sensitive Networking*, as it lays the foundation for the other worksheets. Each chapter throughout this document is its own worksheets and one should therefore be able to read each worksheet individually without necessarily reading the others. Each worksheet can include multiple sections supporting the giving worksheet.
Test and test-journals are presented towards the end of each worksheet, as some theory might be necessary to better understand the different test.

Citations is marked in square-brackets with a number corresponding to a source in the bibliography, e.g. [1].

Authors :

- Andreas Engelsen Fink
- Davide Ganna
- Micheal Otieno Okumu
- René Dam Marcker
- Søren Aaberg Markussen
- Taus Mortensen Raunholt

Contents

Chapter 1

802.1Q Time-Sensitive Networking

Time-Sensitive Networking (TSN) is an enhancement of the IEEE 802.1 and 802.3 standards [1] and sets the foundation for communication in future factories. This was enabled by the addition of a Virtual Local Area Network (VLAN) tag to the Ethernet header as illustrated in figure 1.1. The components found in the VLAN tag and its functionality is illustrated in table 1.1. The addition of the tag enables more functionalities and capabilities as compared to the standard Ethernet. The functionalities includes e.g. traffic shaping mechanisms, bandwidth allocation, frame prioritisation and other queue management techniques. These new functionalities and capabilities can be applied to the increasingly demanding industry. Table 1.2 depicts the various enhanced standards which can be used in future factories to solve certain network demands. These features could not be implemented with the traditional Ethernet standard, resulting in the need for introducing a new standard which has focus on industrial utilities.

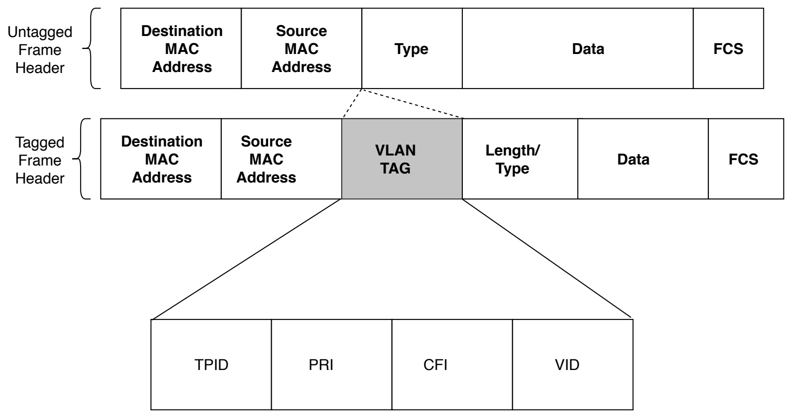


Figure 1.1: Illustrates the VLAN tag and its components added to the traditional Ethernet frame. The un-tagged frame illustrates the original ethernet frame before the introduction of the TSN standard

Table 1.1: Components of the VLAN tag and their functionalities.

Field		Length in bits	Description	Value
TPID		16	Tag Protocol Identifier (TPID) indicating the frame type.	Field set to a value of 0x8100 in order to identify the frame as an IEEE 802.1Q-tagged frame. Device vendors can define their own TPID values, and users can then change the value to realise interconnection of devices from different vendors.
TCI	PCP	3	Priority code point (PCP) bits indicates the priority of the frame.	Different PCP values can be used to prioritise different classes of traffic. The values can range from 0 to 7, with the largest value giving a higher priority. If Strict Priority Scheduling is enabled, the packet with highest priority will be sent first. If multiple packets contain the same priority, a FIFO mechanism will be used among this priority.
	DEI	1	The Drop Eligible Indicator (DEI) field. This may be used separately or in conjunction with PCP to indicate if frames are eligible to be dropped in the presence of congestion.	It has value range of 0 to 1, with 0 being default. A value of 1 indicates drop eligibility, and that the packet can be dropped if necessary.
	VID	12	VLAN ID (VID) field specifying the VLAN to which a frame belongs.	The VLAN ID is in the range from 0 to 4095. It allows the sub-division of a physical network into separate logical broadcast domains.

Some of the benefits of the new TSN standard [1] as a result of the tagging are:

- Flexibility
- Reliable delivery of packet
- Ability to converge application and traffic on a single open network
- Guaranteed latencies
- Ease of network configuration and management

The addition of the VLAN tag has resulted in several 802.1Q enhancements, whereby some sub-standards have emerged. A select few of these are illustrated in table 1.2.

Table 1.2: Some of the IEEE 802.1Q enhancement standard [1]

Standard	**Area of Definition**	**Title of Standard**
IEEE 802.1AS, IEEE 1588	Timing and Synchronisation	Timing and Synchronisation
IEEE 802.1Qbv	Forwarding and Queuing	Enhancements for Scheduled Traffic
IEEE 802.1Qbu	Forwarding and Queuing	Frame Preemption
IEEE 802.1Qav	Forwarding and Queuing	Forwarding and Queuing Enhancements for Time-Sensitive Streams
IEEE 802.1Qci	Time-based Ingress Policing	Per-stream Filtering and Policing

The table illustrates some of the IEEE 802.1Q enhancements which will be elaborated in the different worksheets. In the enhancements, there are features which have contributed to the realisation of the Industry 4.0 These features allow more reliable transmissions of both critical and non-critical data over the same medium. This is achieved through, among others, the availability of mechanisms such as traffic shaper (802.1Qbv), which guarantees the data delivery using Ethernet-based communication [1]. With the TSN standard, there are three data delivery guarantees which can be termed as follows [1]:

- Latency: data delivery of each packet in a stream is guaranteed to occur at the receiver within a predictable time,
- Bandwidth: data delivery of each packet in a stream is guaranteed to occur at the receiver provided that the bandwidth is sufficient for the reserved sender,
- Deadline: data delivery of the packets in a stream is guaranteed to arrive at the receiver within a predefined time frame.

The TSN standard addresses the issues of critical data. This data criticality can be used to guide the selection of the appropriate QoS in case of conflicting requirements. The standard defines the following criticality: [1, Table 2]

- High: used by highly critical network services or application to the system. For instance network control features such as clock synchronisation are considered to be critical to the operation of the network,
- Medium: used by relevant, but not necessarily required, network services for the operation of the critical parts of the system, e.g. alarms or operator commands which may tolerate a small delay of up to 2 seconds.
- Low: used by network services or applications which are not relevant for the operation of the critical part of the system, e.g. video surveillance traffic used for monitoring production.

1.1 Prioritisation in TSN

The use of the 802.1Q standard allows the configuration of up to 8 priorities, as shown in table 1.1. The distribution of these priorities are however not determined by the standard, i.e. for a given network one may choose which traffic types fall under the different priorities. Nonetheless, instead of using a vast range of different configurations and adapting to these, following a specific set of guidelines can be beneficial.

One such recommendation has been created by the Industrial Internet Consortium for their Time-Sensitive Networks for Flexible Manufacturing testbed [1]. This recommendation is based on the QoS requirements of various traffic types, and takes into account the distribution of the data as well as the deadlines of the traffic. Furthermore, the guideline also states whether the TSN sub-standards should be used, e.g. if the 802.1Qbv shaper is necessary to reserve bandwidth for high-priority traffic.

A list of selected recommendations can be seen in table 1.3. These guidelines will not be used for the various tests in the project, but instead provides insight as to what should be considered when configuring a network.

Table 1.3: Selected recommendations for traffic shaping from the Industrial Internet Consortium [1, Table 14].

Types	**Priority**	**802.1Qbv**	**802.1As**	**802.1Qbu**	**802.1Qav**
Network Control	7			C	
Isochronous Traffic	6	M	M		
Cyclic	5	M	M	R	
Events - Control	4			O	
Events - Alarms & Operator Commands	3			O	O
Configuration & Diagnosis	2			O	
Video, Audio, Voice	1			O	R
Best Effort	0			O	

Legend:

- M: Mandatory
- O: Optional
- C: Conditional
- R: Rate-based

Chapter 2

Clock Synchronisation

Although computers and other devices all contain clocks and are therefore able to tell the time down to the nanosecond, these clocks do not necessarily tell the same time across multiple devices. In order to properly communicate within a network, these clocks need to be synchronised to some central clock either inside the network or a dedicated server hosted elsewhere. Although having synchronised clocks is not necessary for a successful transmission, it is important for e.g. periodic traffic, and especially for time-sensitive networks. If the clocks are not synchronised, the various clock drifts may eventually cause errors.

The purpose of clock synchronisation is therefore to provide the network with an accurate and reliable time synchronisation. One implementation of synchronisation is defined in the standard IEEE 802.1AS [2].

2.1 Precision Time Protocol

The Precision Time Protocol (PTPv1) is a time synchronisation protocol that is used in a network to synchronise the clocks of different devices to a single clock. This works by designating one of the clocks on the network as the Grand Master (GM). The clock of the GM will determine the clocks for the entire network, such that the clocks of the other devices (referred to as 'slaves') will be synchronised to it. To determine which clock will give the most precise and consistent time, and thus become the GM, the devices in the network will use the Best Master Clock (BMC) algorithm [3]. This algorithm will be presented later in this worksheet.

For a slave clock to be synchronised with a GM, the GM will send a sync message to the slave clock, followed by a sync follow up message that contains the timestamp for when the sync message was sent from the GM. The slave clock then logs both when the first message was received (T2) and when it was sent from the GM (T1). It then replies to the GM by transmitting a delay request message and log the time it was sent (T3). When the GM has received the message, it will be registered (T4) and transmitted back the slave clock [3]. The slave clock can then calculate the time difference between the two clocks using the formula 2.1, which is the one-way offset [3].

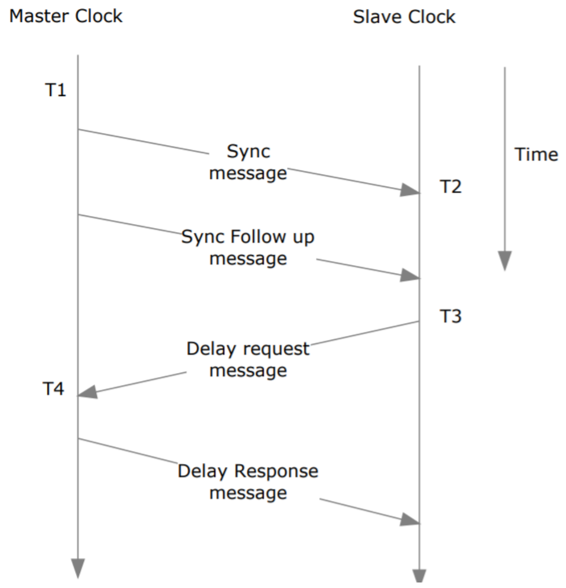


Figure 2.1: Illustration of the PTP synchronisation process [3].

$$P_{delay} = \frac{(T_2 - T_1) - (T_4 - T_3)}{2} \tag{2.1}$$

An illustration of the PTP synchronisation process can be seen in figure 2.1.

Best Master Clock

To determine the best clock, a network device will transmit a packet with information about its clock, as well as other identifying information [4]. This packet will be compared by other systems to their own information, and the most accurate clock is set as the GM. The criteria to become the BMC consists of 6 points, which work as a prioritised list [4], which can be seen below:

1. Priority Field: This field is normally set to a value of 128 for contesting devices (255 to specify slave only devices) but is otherwise used to manually set the GM. This enables the possibility to specify e.g. fall-back devices in case the optimal GM is disconnected unexpectedly.
2. Clock Class: The clock that has the best external synchronisation, i.e. if a clock has GPS synchronisation it gets a higher priority than a clock that does not synchronise to any outside sources.

3. Clock Accuracy: Drift compared to UTC.
4. Clock Variance: Statistics on the jitter and oscillator drift of a clock.
5. Priority 2 Field: Manually set field to determine primary and backup clocks.
6. Source Port ID: Unique port ID to determine tiebreakers.

If a clock has a better value in the priority field, it will be selected as GM. If none of the clocks are manually set as the master, the devices will go down the list and check the next priority on the list. They will then check who is the most precise of the clocks, if one of the clocks are more precise than the other it will be selected as GM. As this list is used to determine which clock is most accurate, each point of the list is checked. If two clocks are identical, properties not related to the performance of the clocks will be used as tiebreaker [4].

When a clock connects to a network
As a system connects to a network it will start by listening for at GM signal. If it receives a signal containing information about a better clock it becomes a slave, and will be synchronised to the GM. If a better clock is not found within a preset period of time, the clock will assume the role of GM. If two or more clocks are connected to the same network and they do not receive a GM signal, they will both try and assume the role of GM. To determine which clock is better, the two will exchange information about their precision through the BMC algorithm. The best clock will be chosen and take the role of GM for the system [4]. This process can be seen in figure 2.2. Devices in a network will follow this state diagram indefinitely, i.e. if the GM is suddenly disconnected, then the BMC algorithm is used again to determine a new GM.

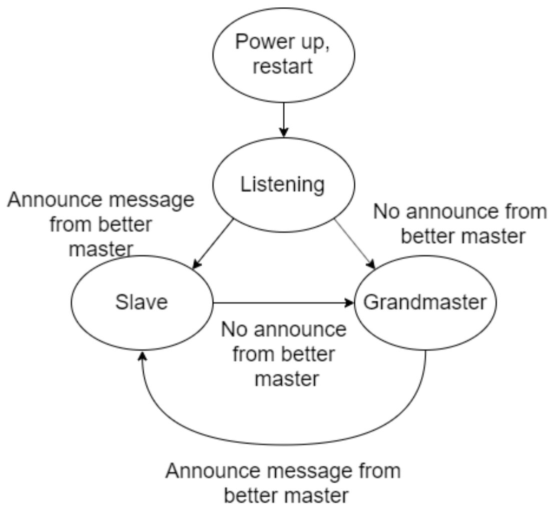

Power up, restart
Listening
Announce message from better master
No announce from better master
Slave
Grandmaster
No announce from better master
Announce message from better master

Figure 2.2: State diagram of the BMC algorithm.

What is described throughout this section is in principle how PTP works. It should be noted that many other variations or profiles of the PTP protocol have been created. Most profiles are however built upon the original PTP protocol [5].

2.2 Generalised Precision Time Protocol

Generalised Precision Time Protocol (gPTP) uses the IEEE 802.1AS profile (802.1AS), which is based on IEEE 1588-2008, also known as PTP, and is specified by the Time-Sensitive Networking (TSN) task group [6]. The synchronisation of gPTP is as for PTP described in the previous section 2.1, which is built around the master-slave principle. gPTP is however more robust against jitter caused by e.g. buffers in switches. This is because it is necessary for all devices in a network to support MAC-level PTP in order to use gPTP [6]. As a result, the propagation delay is lowered due to the PTP-packets in the network being prioritised higher than other packets.

To measure the propagation delay between two gPTP capable end-to-end devices P1 and P2, a packet delay request (Pdelay_req) message is sent from P1 with its local time t1. When the Pdelay_req message is received at P2, its own local time is recorded at t2. At P2's local time (t3) a response, Pdelay_resp, message is sent to P1. When P1 receives the response from P2, its local time is recorded, t4. As a final step, P2 sends a follow-up message to P1 containing the departure time, t3. This process is illustrated in Figure 2.3. At the end of this process, P1 knows all necessary timestamps to calculate the time offset as shown in equation 2.2.

$$P_{delay} = \frac{(t_4 - t_1) - r(t_3 - t_2)}{2} \tag{2.2}$$

To account for the local clock frequency variation at P2 compared to P1, also known as clock drift, we multiply the time interval (t3-t2) with a factor r which is the ratio of the clock drift between the master and the slave. To determine r, two messages are sent from the master to the slave, which is illustrated in figure 2.3(b). Because of the clock drift, we know that the time interval (t12 - t11), which is the departure time, is not equal to the arrival time time interval (t22 - t21). The ratio r can be calculated as the ratio of these intervals seen in equation 2.3.

$$r = \frac{t_{12} - t_{11}}{t_{22} - t_{21}} \tag{2.3}$$

If the ratio is equal to or very close to one, the clock synchronisation works as intended, since the local time at each of the systems are identical. If r is less than one, the master clock is behind the slave and vice versa if r is greater than one.

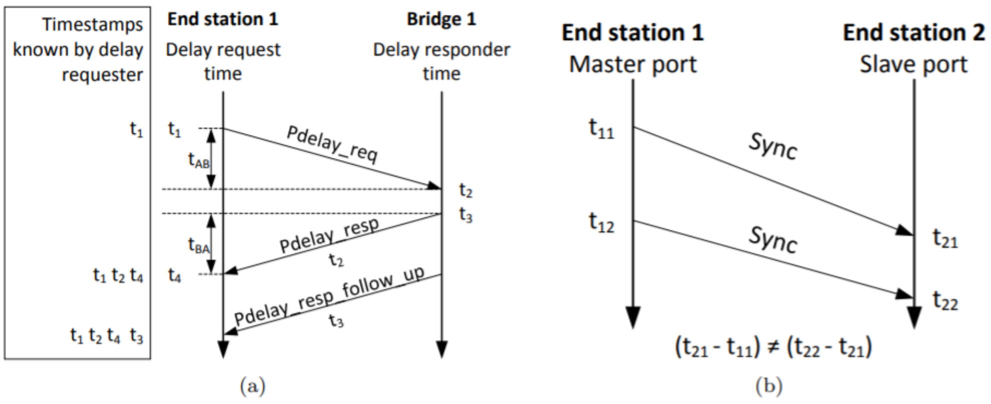


Figure 2.3: In figure (a) the sequence diagram shows the measurement of the propagation delay. Figure (b) the sequence diagram shows the measurement of the clock drift on the arrival time of the messages. [6]

For this project, gPTP will be used, as it builds upon the original PTP protocol, however with focus on time-sensitive networking.

2.3 Test Journal - gPTP accuracy

Purpose of the Test

Parts of the 802.1Q time-sensitive networking protocol relies on high reliability in terms of clock synchronisation, and it is thus relevant to look into the performance of the gPTP of the system. This test will be used to determine the precision of the synchronisation protocol in conditions when there is no other traffic than gPTP, and when high amounts of best effort traffic is introduced. Furthermore, implementing the prioritisation of 802.1Q, known as Strict Priority Scheduling (SPS), and specifying the PTP traffic as having a higher priority than the best effort traffic should result in improved accuracy compared to using no such features.

Theory

This test uses the gPTP protocol, which works as described in section 2.2. The SPS algorithm is presented in the worksheet regarding 802.1Q, see chapter 1.

Test setup

Two computers connected using a single 1 Gbit/s Ethernet cable will be used for this test. Both computers are equipped with the Intel I210 network card, with one computer taking the role of grand master (GM), and thus providing the synchronisation service, while the other computer takes the role of slave. This means that the BMC algorithm will only be used up until the first point, in which a priority setting is checked. Since the network only consists of two end-points of similar hardware, not determining the best master using clock performance will not have any significant impact on the performance.

The implementation of gPTP used for this test is provided with the TSN tools provided by OpenAvnu [7]. Furthermore, in order to use the tools, sample applications provided by Intel will be used to simplify the initialisation [8].

In order to start the synchronisation protocol on the computer acting as GM, the command shown in listing 2.1 is used. The slave is likewise configured using the same command, with *boardA* changed to *boardB*. Note that *iotg_tsn_ref_sw* specifies the directory of the files from the sample applications provided by Intel.

Listing 2.1: gPTP command on machine acting as GM. In this case, the interface *enp1s0.3* is used.

```
$ cd /iotg_tsn_ref_sw/scripts/
$ sudo bash setup_sync.sh -i enp1s0.3 -b boardA
```

These commands will be used on both computers with the only difference being "boardA" changing to "boardB" on the slave PC. The commands will run tools from the OpenAvnu, and initialise the relevant programs for the protocol to function. The output of the gPTP tool is shown below, with the tool being referenced as *ptp4l*.

Listing 2.2: Output for machine acting as GM.

```
ptp4l[61.760]: selected /dev/ptp1 as PTP clock
ptp4l[61.862]: port 1: INITIALIZING to LISTENING on  INIT_COMPLETE
ptp4l[61.862]: port 0: INITIALIZING to LISTENING on  INIT_COMPLETE
ptp4l[68.762]: port 1: LISTENING to MASTER on
ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[68.762]: selected local clock 6805ca.fffe.a7a06e as best
master
ptp4l[68.762]: assuming the grand master role
```

Listing 2.3: Output for machine acting as slave. The Offset, in nano seconds, compared to the master is marked in red.

```
ptp4l[39.123]: selected /dev/ptp1 as PTP clock
ptp4l[39.246]: driver changed our HWTSTAMP options
ptp4l[39.246]: tx_type   1 not 1
ptp4l[39.246]: rx_filter 1 not 12
ptp4l[39.246]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[39.246]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[39.247]: port 1: link up
ptp4l[39.714]: port 1: new foreign master 6805ca.fffe.a7a06e-1
ptp4l[43.715]: selected best master clock 6805ca.fffe.a7a06e
ptp4l[43.715]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[44.715]: master offset -4215601 s0 freq  -544 path delay 0
ptp4l[45.715]: master offset -4215593 s1 freq  -536 path delay 0
ptp4l[46.715]: master offset    -4568 s2 freq -5104 path delay 0
ptp4l[46.716]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[47.716]: master offset       -2 s2 freq -1908 path delay 0
ptp4l[48.716]: master offset     1015 s2 freq  -892 path delay 357
ptp4l[49.716]: master offset     1325 s2 freq  -278 path delay 404
ptp4l[50.716]: master offset     1066 s2 freq  -139 path delay 404
ptp4l[51.717]: master offset      622 s2 freq  -263 path delay 452
ptp4l[52.717]: master offset      232 s2 freq  -467 path delay 561
ptp4l[53.717]: master offset       73 s2 freq  -556 path delay 649
ptp4l[54.717]: master offset      113 s2 freq  -494 path delay 622
ptp4l[55.717]: master offset       63 s2 freq  -510 path delay 622
ptp4l[56.718]: master offset       23 s2 freq  -531 path delay 628
ptp4l[57.718]: master offset       14 s2 freq  -533 path delay 624
ptp4l[58.718]: master offset        3 s2 freq  -540 path delay 624
ptp4l[59.719]: master offset        8 s2 freq  -534 path delay 623
ptp4l[60.719]: master offset        2 s2 freq  -542 path delay 623
```

As shown on listing 2.2, the system will take the role of GM after a few seconds, using its own local clock. The slave, shown in listing 2.3, will meanwhile find the GM on the network, and thus start calibrating its own clock to it. Information is also provided as output, in which the calibrated offset in nanoseconds, the offset between the two clocks in parts per million (ppm)(marked with red), and the path delay in nanoseconds. In order to determine the accuracy over a longer period of time, the offset provided in the slave output is saved.

The same test can be conducted with traffic present, in which approximately 950 Mbit/s of best-effort background traffic generated using iperf3 is introduced. This is expected to cause congestion on the connection, causing delays or potentially packet drops for the PTP packets. Finally, a test using the 802.1Q prioritisation can be conducted, where PTP packets has a specified priority higher than best-effort traffic.

To enable the 802.1Q prioritisation, the command in listing 2.4 was executed. This is a generic linux command, but uses a configuration specified by OpenAvnu. Although the specifics of this command will not be explained in this worksheet, the result of the command is as follows:

- The initial FIFO setup is replaced with the 802.1Q prioritisation setup
- gPTP packets, which has a priority of 7, will be mapped to queue 0. Remaining packets will be mapped to queue 3.
- No shapers are applied, i.e. the priority is the only factor used to determine which packet should be transmitted. In the case that multiple packets of the same priority are present, FIFO is used among those.

Listing 2.4: Command for specifying the 802.1Q priority.

```
$ sudo tc qdisc replace dev enp1s0 parent root handle 100    \
  mqprio num_tc 4 map 3 3 3 3 3 3 3 0 3 3 3 3 3 3 3 3        \
  queues 1@0 1@1 1@2 1@3 hw 0
```

Results

The tests were conducted in a period of roughly 13-16 hours, in which the slave would synchronise to the GM once every second. This has resulted in 129,723 measurements when the medium was idle, 158,564 measurements when traffic of approximately 950 Mbit/s was present, and 158,257 with traffic and using the 802.1Q prioritisation.

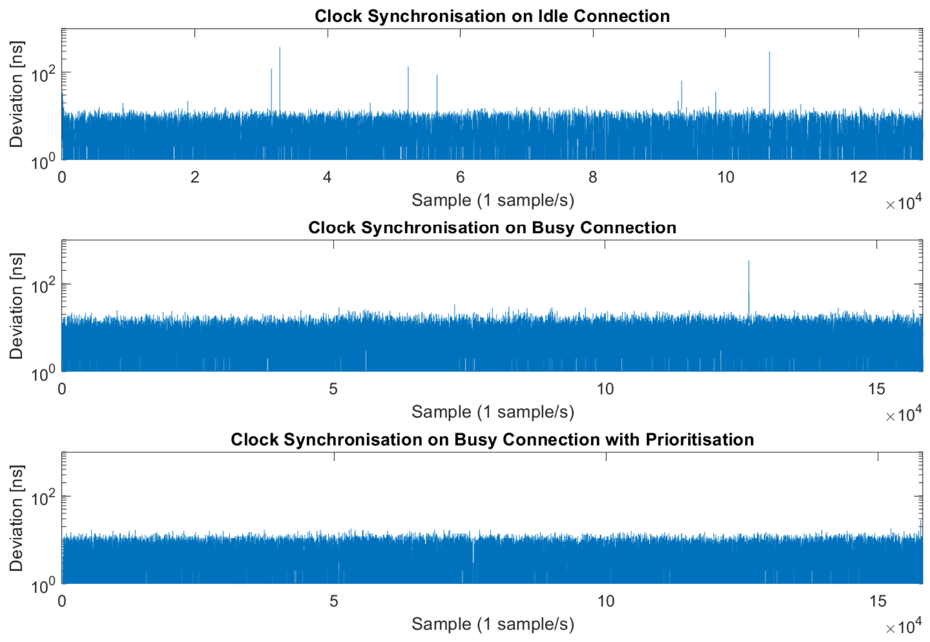


Figure 2.4: Slave clock offset from GM. The slave synchronises at a rate of one correction per second.

Figure 2.4 shows the offset over time period when the medium is idle and when traffic is present. Note that the initial 20 samples have been removed, as the slave was being initially synchronised during this time, and does not properly reflect the performance of the gPTP in general. The removed samples corresponds to a duration of 20 seconds.

The spikes seen on the figures are a result of the slave losing connection to the GM. Since the interface used in the test (Ethernet) is believed to be sufficiently reliable to not cause any such errors, it is believed that this is caused by errors in software. It should however be noted that the most errors occurred when no traffic was present, meaning that the errors were not caused by congestion. Nonetheless, the errors resulted in an offset of at most 376 ns. The clock of the slave would likewise remain unsynchronised for a maximum period of 21 seconds, before the synchronisation protocol was reestablished. This indicates that when a GM disconnects from a network, the clock will remain relatively accurate until a new GM is decided upon.

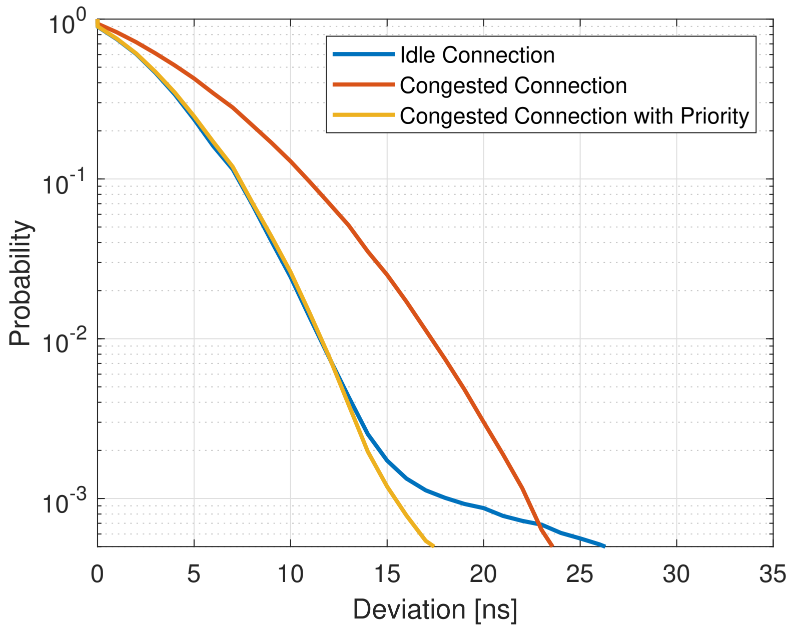


Figure 2.5: CCDF of slave clock offset under various conditions.

Table 2.1: Selected statistics of the clock offset under various conditions.

	Min	**Median**	**99%-ile**	**99.9%-ile**	**Max**	σ^2
Idle	0 ns	3 ns	12 ns	21 ns	376 ns	13.80
Congested	0 ns	5 ns	18 ns	23 ns	337 ns	18.38
Congested w/ Priority	0 ns	3 ns	12 ns	16 ns	92 ns	8.52

Figure 2.5 and table 2.1 shows a CCDF of the offset for the three tests. It is here seen that using a congested interface will result in a larger offset, although only in terms of a few nanoseconds. When 802.1Q packet prioritisation is used, however, the offset achieves the same accuracy as when no traffic is present. This is reasonable, as the packets will spend very little time in the buffers before being transmitted, as opposed to waiting until an entire buffer is sent.

It can also be seen on the figure that in the scenario where prioritisation is used, the accuracy is seemingly better than when no traffic is present. Since the two conditions are in principle the same, as packets are sent near-instantly, the difference in time is small enough that the two can be considered to be identical. If more datapoints were captured, it would have been possible to conclude upon this with a higher degree of certainty.

Clock synchronisation - conclusion

The clock synchronisation has been found to have an offset of at most 18 ns for 99.9%-ile of the samples when the medium is idle, and similarly 23 ns for when traffic is present. When traffic is present and prioritisation is used, an offset of at most 16 ns for the same amount of measurements is observed. The decreased accuracy can however be mitigated by reserving the medium when gPTP packets needs to be transmitted by utilising the SPS algorithm. This will result in performance identical to when no traffic is present.

Although the role of GM was assigned beforehand, the accuracy is maintained when used in a congested medium, indicating high robustness and sub 1 µs accuracy with the given test setup.

Chapter 3

802.1Qav - Credit Based Shaper

Current industrial communication technologies are composed by a large amount of low priority traffic, and only occasional high priority traffic bursts [1]. The system will react differently depending on the protocols used, but the devices will stop the transmission of low priority traffic if a non-optimised protocol is used. It blocks the low priory to allow the high priority packets to be transmitted. Alternatively, the system could use a scheduling protocol to ensure that both high and low priority traffic can be transmitted at the same time.

In the case of a non-optimised protocol a high bursts of data will occur, which may not be beneficial for the network. If a scheduling protocol is used, the system will reduce the amount of low priority messages that can be transmitted and could in turn avoid congestion together with unnecessary bursts in the network.

One can think that this problem can be solved by adding bandwidth to the link, but this does not completely eliminate the potential for network congestion. In fact, there will always be critical points in the network where multiple traffic streams merge or where network links change in terms of speed and capacity. Furthermore, the impact and number of these congestion points will increase over time as more applications and devices are added to the network [9].

A third way to transmit high and low priority traffic on the same link is to limit the amount of bandwidth of a high priority transmitter, in order to allow the desired low priority traffic to be sent without interfering. This is done using the IEEE 802.1Qav Credit Based Shaper (CBS).

The 802.1Qav shaper is a Credit Based Shaper (CBS)[10], meaning it utilises credits to determine the throughput, as shown in figure 3.1. The amount of available credits will in general rest at 0, and will increase under two conditions:

- Shaped frames are ready to be transmitted, but the medium is blocked due to interference, resulting in a positive amount of credits being accumulated until an upper limit is reached, named the Hi-credit.
- When the amount of credits is negative, up until the resting point at 0.

The rate at which credits are gained is determined by a parameter named the Idleslope.

When a frame is being transmitted, credits are used based on the length of the frame in bytes and can result in negative credits. The rate at which credits are consumed is determined using the Sendslope parameter. Similar to gaining credits, there exists a lower bound denoted by Lo-Credit.

In the case of an idle network, this will result in the shaped traffic to be transmitted periodically. After a frame is transmitted, the credits has to be restored to 0, after which another frame is sent and the credits reach negative values again. If interference is present while frames are available to be transmitted, the positive amount of credits allows for some bursts when the medium is freed again. These burst will result in the overall throughput reaching the target rate.

When using the 802.1Qav shaper in conjunction with the Strict Priority Scheduling (SPS), presented in chapter 1, it is possible to limit the throughput of the shaped traffic to allow for some lower priority transmissions. If no traffic of higher priority than the shaped traffic is present, no interference will be present for the 802.1Qav shaper, and the transmissions will follow a periodic pattern. The best effort traffic, which has lower priority, can then be transmitted inbetween the shaped traffic.

The following figure 3.1 illustrates this concept:

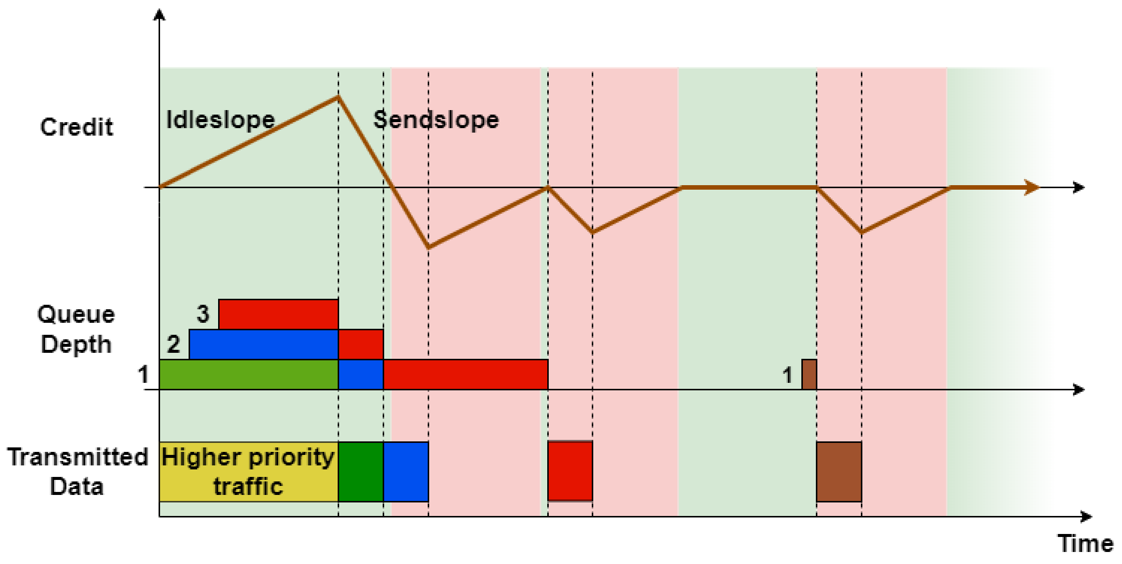


Figure 3.1: 802.1Qav Traffic shaping [11].

From the figure, it can be seen that while high priority traffic (in yellow) is currently transmitted, three different frames are queued. When there is at least one frame in the queue, credits are accumulated. Once the high priority traffic has been transmitted, the packets in the queue start to be transmitted with a FIFO mechanism. The third packet (in red) will not be sent until the available credits are non negative. If no packets are present in the queue, the credits will stabilise at a value of zero. For this reason, the fourth packet (in brown) will be instantly transmitted as soon as it enters the queue.

3.1 IEEE 802.1Qav Implementation

The Linux implementation of CBS is done at the kernel level, using the traffic control package. when creating a CBS it is necessary to use the queuing discipline or also known as qdisc. This allows the system to implement the CBS on the Network Interface Card (NIC). Along with the qdisc command, a multitude of arguments are needed to set the the value for the increase/decrease in credits and how high/low the credits can go.

An upper and lower bound can be set for the maximum/minimum amount of stored credits: *Hicredit* and *Locredit* are the arguments needed to set these. *Hicredit* can be used to limit the maximum burst allowed by a system after longer periods of interfering traffic, while *Locredit* is used to minimise the largest period the system has to wait until a new packet can be sent.

The testbench is configured using network tools from OpenAvnu [7], as well as the configuration tools and sample programs by Intel [8].

Implementation of CBS

The command to implement a CBS policy is shown in listing 3.1.

Listing 3.1: Command to implement the CBS on a queue. Here, the interface *enp5s0* is used, and the queue located at 100:1 is configured.

```
sudo tc qdisc replace dev enp5s0 parent 100:1 cbs idleslope 7808
sendslope −992192 hicredit 12 locredit −97
```

Where:

- Idleslope

  Idleslope is the rate of credits that are accumulated (in kilobits per second) when there is at least one packet waiting for transmission. This is the main tunable parameter of the CBS algorithm and represents the bandwidth that will be consumed.

- Sendslope

Sendslope is the rate of credits that are depleted when a transmission is occurring, this should be negative number in kilobits per second. It can be calculated as follows:

$$Sendslope = Idleslope - Port\_Transmission\_Rate$$

- Hicredit

  Hicredit defines the maximum amount of credits (in bytes) that can be accumulated.

$$Hicredit = Max\_Interference\_Size \cdot \frac{Idleslope}{Port\_Transmission\_Rate}$$

- Locredit

  Locredit is the minimum amount of credits that can be reached. It is a function of the traffic flowing through this qdisc:

$$Locredit = Max\_Frame\_Size \cdot \frac{Sendslope}{Port\_Transmission\_Rate}$$

- Offload

  When offload is set to 1, the network interface card is configured so that the CBS algorithm runs in the network controller. The default is 0, resulting in the CBS being implemented in software only.

3.2 Test Journal - Comparison between Unshaped and 802.1Qav Shaped Traffic

In this section a data analysis of the IEEE 802.1Qav traffic type is provided, along with considerations of the results.

Purpose of the Test

The purpose of this test is to show the difference between the throughput of unshaped traffic compared to when using the 802.1Qav Credit Based Shaper. The raw data will be analysed and plotted graphically: this way the reader can visually see the difference between the two types of queuing mechanisms and have a better understanding of how this shaper works.

Theory

In order to show the behaviour of this particular shaper, the two identical types of traffic has been analysed. In the first case it is treated as a normal interfering traffic (unshaped), whereas in the second case it will be shaped through the CBS.
The functionality of the CBS has been described in the beginning of chapter 3.

Test setup

This test follows the example provided by the Intel IoTG group (Qav Demo 2 Scenario 2) [8], and uses the tools provided by them. The physical setup consists of two computers equipped with the Intel I210 network card. An Ethernet cable is connected directly between the two computers.

On the receiver-computer, a listener program is started, as shown in listing 3.2. Likewise, tcpdump is run to log the data. Afterwards, a new terminal is opened to start an iperf3 server to receive best effort traffic, as shown in listing 3.3.

Listing 3.2: Command to start the listener program on the receiver-computer. In this case, *enp1s0* is used as the interface, and a file is saved at the specified location.

```
$ mrpd -mvs -i enp1s0 &
$ simple_listener -i enp1s0 -f filename.wav
```

Listing 3.3: Command to start an iperf3 server for best-effort traffic.

```
$ iperf3 -s
```

On the transmitter-computer, three terminals are opened and the commands shown in listing 3.4 is run. The MRPD service originates from the 802.1Qat Stream Reservation Protocol standard, and is necessary for this test to be conducted. This protocols allocates the bandwidth along the path between the transmitter and the receiver, but as this is a direct link, it is believed to cause minimal impact. Further investigations of how this protocol works will not be considered for this test.

Listing 3.4: Commands to prepare the transmitter computer. Here, *enp1s0* is the interface of the NIC. The IP address specified for terminal 3 belongs to the receiver-computer.

```
Terminal 1:
$ mrpd -mvs -i enp1s0 &

Terminal 2:
$ ./simple-talker-cmsg -i enp1s0 -t 2 -C 0 -q 1
(Press Ctrl+C and then press Enter afterwards to start)

Terminal 3:
$ iperf3 -c 169.254.0.2 -u -b 150M -l 1448 -t 30
```

After the test has run for 1 minute, all programs are stopped and the data from tcpdump is inspected.

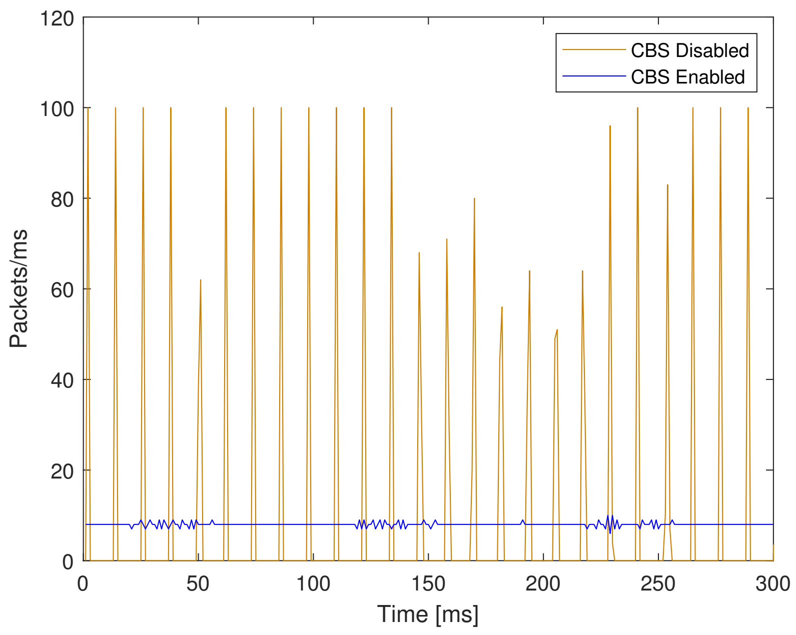


Figure 3.2: Comparison Between Normal Traffic and IEEE 802.1Qav Shaped Traffic.

As seen in figure 3.2, there is a clear difference between the unshaped and shaped traffic. It can also be seen that the shaped traffic do not present any bursts. It is important to emphasise that the traffic transmitted in both cases are the same, so it's legitimate to think that the average number of packets transmitted per unit of time is the same (and indeed it is, as we will shortly see).

An analysis of the data has been performed, leading to the following results:

Table 3.1: Statistics Regarding Unshaped Traffic and Qav Shaped Traffic.

Type of Traffic	**Mean [packets/ms]**	σ^2
Unshaped	8.1	1309
Shaped	8.0	0.339

Since both traffic sources have the same mean in terms of throughput, it is clear that the traffic without CBS has large bursts in short periods of time (i.e. there are moments in which no data is being sent). This can be seen in Figure 3.2, where the peak is at roughly, compared to the shaped traffic of 8 packets/ms. This results in a mean of 8.1 packets every millisecond for the unshaped traffic and 8 packets every millisecond for the shaped one.

Due to the high bursts in the unshaped traffic and the requirement that the mean must be constant over time, a very high variance is experienced in the first case. In contrast to this, the shaped traffic presents only 0.026% of the variance compared to the unshaped one. This results in a complete removal of the previous experienced bursts, leaving the network with a smoother traffic shape.

Conclusion

As seen in the previous sections, shaping the traffic through IEEE 802.1Qav allows a particular traffic to be smoothed, reducing the bursts in the network and thus avoiding possible points of failure. Another important consideration to be taken into account is the amount of packets needed to be transmitted; when using Qav, a lower number of them can be sent compared to the unshaped traffic, due to the constant transmission without any interruption. This allows the network to save bandwidth which may be needed for critical transmission.

3.3 Test Journal - Reconfiguration of the 802.1Qav Shaper

Purpose of the Test

It may prove to be beneficial to change the current configuration of a credit based shaper if the traffic distribution in a network changes. One example of this is adding a new device to a setup while it is running, in which it is necessary to adapt e.g. a specific queue without stopping the entire production. This test aims to determine the impact of reconfiguring the credit based shaper on a single queue. If all communication is stopped during the reconfiguration, it will be necessary to prepare the devices, as time taken to reconfigure the shaper may exceed their deadlines.

Theory

This test uses the 802.1Qav Credit based shaper, as described previously in chapter 3.

Test setup

Two computers connected through a single Ethernet cable will be used for this test. Both computers are equipped with a I210 network card and are running gPTP which is transmitted as proirty 7.

In order to configure the shaper, the previous configuration needs to be reset to default. This is done using the command shown in listing 3.5, which targets the interface *enp1s0*.

Listing 3.5: Command to reset the qdisc configuration to default. In this case, the interface *enp4s0* is used.

```
$ sudo tc qdisc del dev enp1s0 root
```

Table 3.2: Configurations used by the CBS when testing the effect of switching shaper while transmitting data. The Hi- and Lo-credits are given in bytes.

Idleslope	Sendslope	Hi-credit	Lo-credit
7808 kbit/s	-992192 kbit/s	12	-97
21381 kbit/s	-992192 kbit/s	12	-97

For this test two configurations seen in table 3.2 with the difference in the Idleslope will be used. The shaper is configured using the following the commands shown in 3.6. It sets the shaper and configures the first queue with a CBS where the Idleslope set to 7808 kbit/s.

Listing 3.6: Command to configure the qdisc and CBS shaper. When data is transmitted this scribe changes the CBS Idleslope, if activated

```
$ sudo python3 testChangeShaper.py
```

To generate the data two terminals will run the following commands from Iotg:

Listing 3.7: Command of generation priory 0 packets

```
$ sudo ./sample-app-taprio -i enp4s0 -c 169.254.0.2  /
-x 1 -w tsn_prio0.cfg
```

Listing 3.8: Command of generation priory 6 packets

```
$ sudo ./sample-app-taprio -i enp4s0 -c 169.254.0.2  /
-x 1 -w tsn_prio6.cfg
```

3.7 generates udp packets with priory 1, while 3.8 generates udp packets with prioiry 6. The latter will be send to the CBS queue. On the receiver-computer, tcpdump is runned using the following command 3.9 where <filename> is the path to the file which contains all traffic received. The frames will be transmitted using port 4800, and is used as the filter.

Listing 3.9: Tcp commands to gather trasmitted packets

```
$ sudo tcpdump -i enp4s0 -w <FILEPATH>.pcap port 4800
```

When the python program is activated, it will start to switch between the two configurations every 10s.

Results

When changing the CBS while it is transmitting data, it is capable of quickly changing between the two settings without issues, as illustrated in figure 3.3. The NIC utilises less than 13 ms to set the correct shaper, while still transmitting data following the previous settings. In this test the best effort traffic is limited to 150 Mbit/s. As seen in the figure, the traffic flow is not significantly affected by the reconfiguration as no loss in performance or throughput occurred doing tests. This also indicates that the shaper can be toggled, as setting the Sendslope parameter to 0 will result in the queue being effectively unshaped. It is therefore possible to both toggle the usage of the shaper, as well as dynamically change the throughput without any undesired interruptions or bursts.

Figure 3.3 furthermore shows that less than 2 ms are required for the traffic to stabilise, as seen on the slope. This period is however limited by the resolution used in the figure. The inter-arrival times of the individual frames indicate that the actual period is lower, but no exact duration can be given from the gathered data.

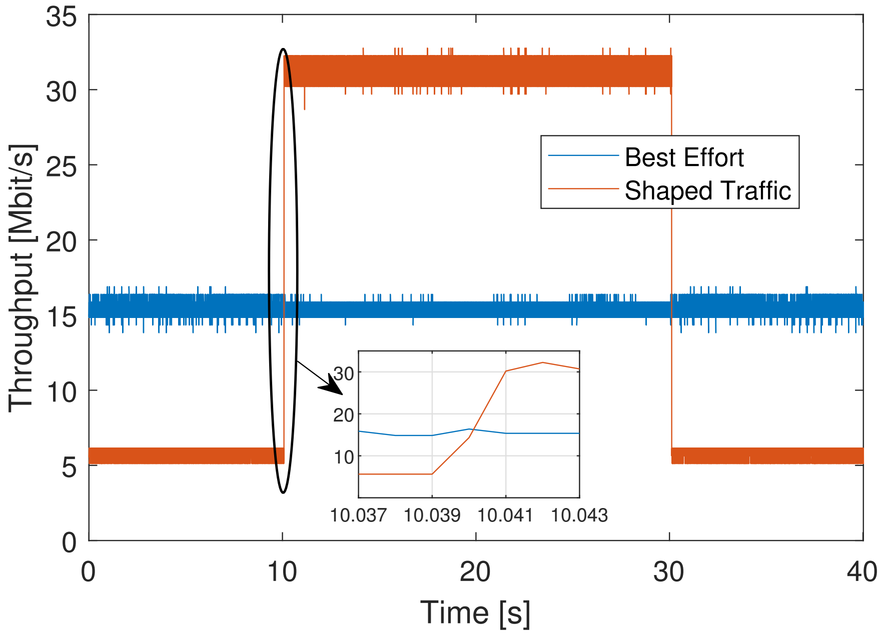


Figure 3.3: The shaper is configured using the default settings with the exception of the Idleslope being changed between the values 7808 and 21381, corresponding to low and high throughput, respectively.

Minimum	Mean	Maximum	Variance
6.2032 ms	8.1911 ms	11.0214 ms	2.6523

Table 3.3: Selected statistics of the CBS shaper reconfiguration time

Reconfiguration of Qav Shaper - Conclusion

Changing configuration during use does not have a negative effect on the traffic, as at most 13 ms was required to process it. Furthermore, as only a single queue needs to be reconfigured, there were no observed interruptions or frame bursts.

Chapter 4

802.1Qbv - Time-Aware Shaper

Time-Sensitive Networking (TSN) emerge as a new standard incorporating real-time network traffic intended for use in industrial scenarios. It is an extension of the Ethernet standard which has many amendments, and among them is the IEEE 802.1Qbv, which will be presented in this chapter [12]. The 802.1Qbv standard defines a time-triggered communication paradigm with the addition of a Time Aware Shaper (TAS). This shaper governs the frame selection at the egress queues according to the predefined schedule, which is encoded in the so-called Gate Control Lists (GCL). This concept is basically a Time-Division Multiple Access (TDMA) scheme. The 802.1Qbv standard aims to guarantee data delivery deadlines as well as guaranteeing minimum latencies and bandwidth [1].

4.1 802.1Qbv Functionality

The 802.1Qbv standard is used in the implementation of the testbench, and it describes TAS. The TAS manages the time controlled gates by blocking all ports except the one expected according to the time schedule in order to avoid delays during scheduled transmission [13]. See figure 4.1 for illustration. The method is a characteristic of the TAS, where frames are not transmitted if the remaining transmission gate open time is not sufficient to ensure the transmission of an entire frame. With this method, the use of guard bands (see worksheet 8), are not necessary to prevent frame interference. One drawback of this however is not fully utilising of the entire transmission gate open time, which can be seen as undesirable in terms of resource utilisation [8]. An illustration of this method can be seen in figure 4.2. The TAS therefore enables determinism by dividing traffic into different time-slots configured in the GCL.

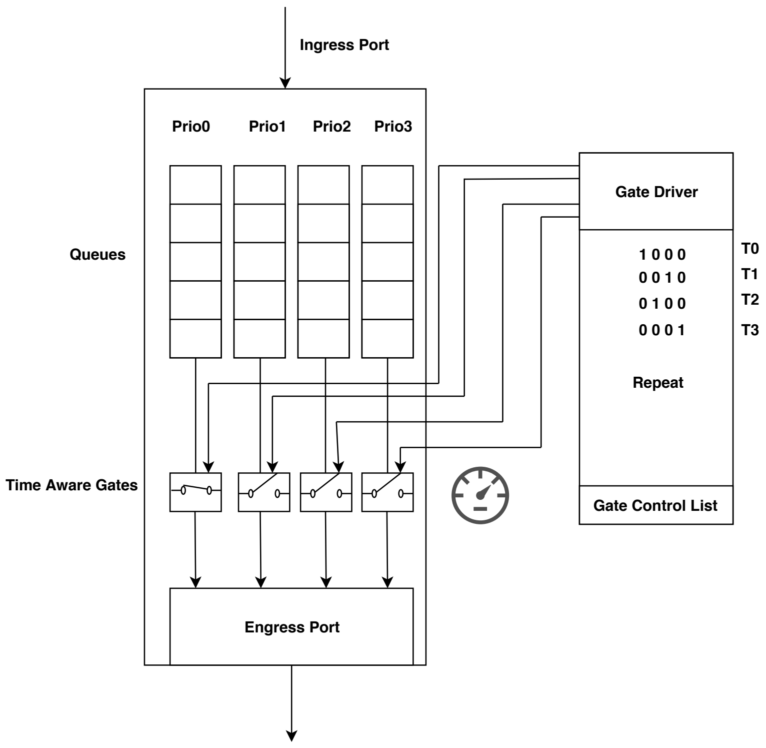


Figure 4.1: A structure of Time Aware Shaper for one Qdisc

If two gates are open in the same time period, the next frame for transmission will be determined from the scheduling algorithm. If SPS is used (see worksheet 1), the frame of highest available priority is chosen for transmission. If multiple frames of the same priority are present, then the First-In First-Out (FIFO) scheduler will be used among them. Frames are selected for transmission on the basis of the priority and how the gate control list (GCL) are set.

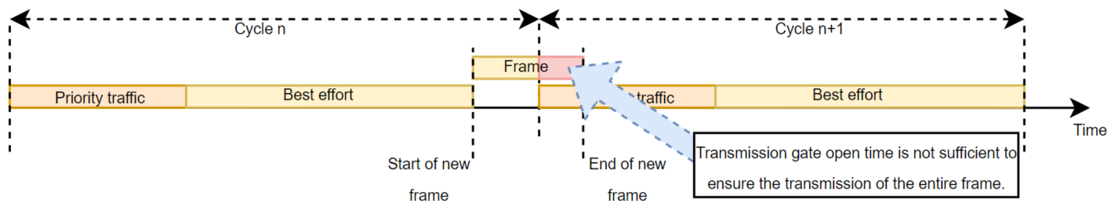


Figure 4.2: An illustration of how the transmission gate open time in this case is not sufficient to ensure the transmission of the entire frame. The frame is therefore not sent until the next applicable window. Best effort (yellow) is seen as the lowest priority and priority traffic (orange) is non-best effort. [14]

Figure 4.1 illustrates how the Qbv is structured. Here it is seen that for every queue there is a Time Aware Gate (TAG). The state of the gate changes with respect to a global time through a predefined Gate Control List (GCL), which is cyclically executed at run-time. In an implementation that does not support improvements for the shaped traffic, all gates are assumed to be permanently in the open state [12, §8.6.8.4]. In order to synchronise the TAG of the two testbenches' Intel I210 network cards, a PTP service is required (see worksheet 2). Note that clock synchronisation between two computers is not necessary in order to apply the TAS for a single transmitter, i.e. only one computer transmitting data. It is however required in the case where e.g. data is requested from a device, and a response is needed within a certain deadline. The synchronised GCL between the two computers will thus mean that the request can be responded to immediately.

The GCL contains the table of how the gates are to be operated, as illustrated in figure 4.3. For any given queue, the transmission gate can be in any of the two states:

– Open: The queued frames will be transmitted according to the defined transmission selection algorithm associated with the queue.

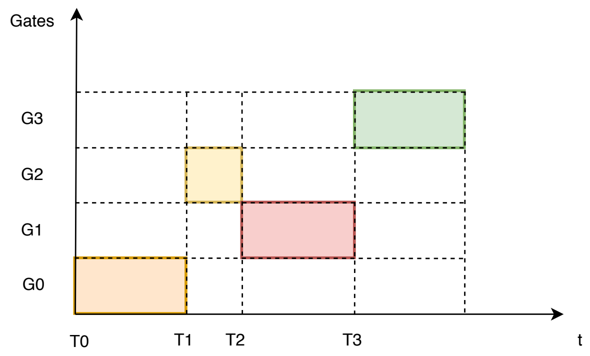
– Closed: The queued frames are not selected for transmission.

Each of these has a time aware shaper (TAS) which controls the four queues as illustrated in figure 4.1. The scheduled events will determine at which instance a queue is opened for traffic to be forwarded to the egress port, and at which time instants the queue is closed such that any pending traffic remains buffered. Each active queue in the Intel I210 represents a single or multiple user priorities.

One of the advantages of GCLs is to provide a periodic bandwidth guarantees, and to also improve the scheduling of the prioritised traffic. This is done by controlling the flow of the frames as illustrated in figure 4.3.

G3	0	0	0	1
G2	0	1	0	0
G1	0	0	1	0
G0	1	0	0	0
	T0	T1	T2	T3

(a) Gate Control List table.



(b) An illustration of the cyclic Gate Control List status at any given time.

Figure 4.3: Illustration of when gates are operated by the time aware shaper.

From the example shown in the figure, it can be seen that the gate G0 is open (1) at the time T0, while the other gates are closed (0). G0, G1 and G3 are closed at time T1 while G2 is open, etc.

The execution of the GCL starts at 'base time' T0, repeats itself after a duration of 'cycle time' has lapsed.

4.2 802.1Qbv Implementation on Testbench

The testbench is configured using network tools from OpenAvnu [7], as well as the configuration tools and sample programs by Intel [8].

The TAS is configured using the command shown in listing 4.1, where the folder *iotg_tsn_ref_sw* contains the tools provided by Intel. This script uses the priority queue configuration shown in listing 4.2, which maps the priority to a given queue. The script applies the provided gate configuration, of which an example is shown in listing 4.3 with a cycle period of 1 ms. For an overview of which gate configuration each hexadecimal value corresponds to, see table 4.1.

Listing 4.1: Script to enable the Qbv shaper on a given interface. In this case, the interface *enp1s0* is used.

```
$ cd /iotg_tsn_ref_sw/sample-app-taprio/
$ sudo python scheduler.py -i enp1s0 -q queue.cfg -e 30          \
   -g gates.sched
```

Listing 4.2: Priority-to-queue mapping configuration. In this example, packets of priority 7 are mapped to queue 0, packets of priority 6 are mapped to queue 1, while the remaining packets of priority 0 through 5 are mapped to queue 3 by default.

```
# [Priority] [Queue]
7       0
6       1
```

Listing 4.3: Gate schedule. The hexadecimal value represents the binary value of all gates. Here, 0x0 represents all gates being closed, while 0xF represents all gates being open.

```
S 0x0 5000
S 0xF 175000
S 0x0 99820000
```

Table 4.1: List of gate control values, where '1' and '0' corresponds to an open and closed gate, respectively.

HEX	Queue state				HEX	Queue state			
	Q3	Q2	Q1	Q0		Q3	Q2	Q1	Q0
0x0	0	0	0	0	**0x8**	1	0	0	0
0x1	0	0	0	1	**0x9**	1	0	0	1
0x2	0	0	1	0	**0xA**	1	0	1	0
0x3	0	0	1	1	**0xB**	1	0	1	1
0x4	0	1	0	0	**0xC**	1	1	0	0
0x5	0	1	0	1	**0xD**	1	1	0	1
0x6	0	1	1	0	**0xE**	1	1	1	0
0x7	0	1	1	1	**0xF**	1	1	1	1

In order to transmit data, the program *sample-app-taprio* provided by Intel is used. The command used to run this program is shown in listing 4.4. This program transmits frames of 64 bytes, matching the minimum size of a frame for Ethernet. After selecting the interface *enp1s0*, the address of the receiver is specified. Following this, the program is set to transmit packets through the x=1 specifier. In order to determine how much traffic should be transmitted, as well as specifying the priority, the file *tsn_window.cfg* is provided, and is shown in listing 4.5. Finally, a base time is specified to synchronise the windows to the Qbv configuration. This time is provided by the script *scheduler.py*.

Listing 4.4: Program to transmit prioritised data.

```
$ cd /iotg_tsn_ref_sw/sample-app-taprio/
$ sudo ./sample-app-taprio -i enp1s0 -c 169.254.0.2 -x 1         \
  -w tsn_window.cfg -B base_time
```

Listing 4.5: Transmitter window configuration. The transmitter is configured with a cycle time of 250 ms, and to transmit 1 packet of priority 6 after the specified offset from when a cycle starts.

```
cycle_time              250000000
priority                6
number_of_windows       1
```

window_1_offset	5000
window_1_duration	1000
window_1_packets	1

4.3 Test Journal - Minimum 802.1Qbv Window Size

Purpose of the Test

Determining the minimum window size for the implementation of the 802.1Qbv protocol on the testbench will provide a lower bound on how strict a schedule can be set, as it may be relevant to allow exclusive access for a single periodic frame. With this test, it will be possible to determine to which degree this window can be minimised, resulting in a reduced idle time for a single packet transmission in a given window.
Note that the focus of this test is to determine the window in which a frame can be transmitted, and not the window in which a pre-queued frame in the network card can be sent.

Theory

The TAS will be used for this test, and has been described in the previous worksheet. See section 4.1 for a description of how to implement the shaper.

Although the transmission window can be specified down to a single nanosecond, as it will be shown in the test setup, it is relevant to first determine what a theoretical minimum frame duration is. Assuming a 1 Gbit/s network link and transmission of a frame consisting of 64 bytes:

$$\frac{64\mathrm{B}}{1\mathrm{Gbit/s}} \approx 0.5\mu s \tag{4.1}$$

Note that this is not considering any buffering or other types of processing, and that the frame is put on the line immediately as its respective window is open. For the implementation on the testbench, the frame will first have to be moved from its queue to an output buffer, and then transmitted.

Before a frame is transmitted, the network card checks the length of a frame and estimates if it is feasible to transmit it within the remaining part of the window. If this is not the case, the frame will not be transmitted at that given time, and will instead be buffered for the next applicable window. This feature will also have an impact on the minimum window size, while also providing a guarantee that the transmissions in each window will not affect other neighbouring windows.

Test setup

Two computers connected through a single Ethernet cable will be used for this test, as to avoid additional traffic in the network. One computer will be used to transmit UDP packets, while the other will log the received packets using tcpdump.

The TAS will be implemented as described in section 4.2. It will be configured to use the schedule shown in fig 4.4, with x being a variable size of the transmission window. This schedule will result in a cycle time of 100 ms, or 10 frame transmission windows per second. The window of size a is set to a fixed value of 5 µs, as an offset from the start of a cycle is necessary for the transmitter-program to function.

In order to determine if a given window size provides reliable transmissions, the following procedure will be used.

1. Set the transmission window to x nanoseconds.
2. Run *scheduler.py*, followed by *sample-app-taprio*.
3. Log data for 1 minute.
4. If exactly 1 frame is transmitted per 100ms, the window size is considered reliable.

This procedure will be repeated for smaller windows, until a minimum window size is found. An illustration of the schedule is shown in figure 4.4, where a will be set to 5 µs, and a constant cycle time of 100 ms is used.

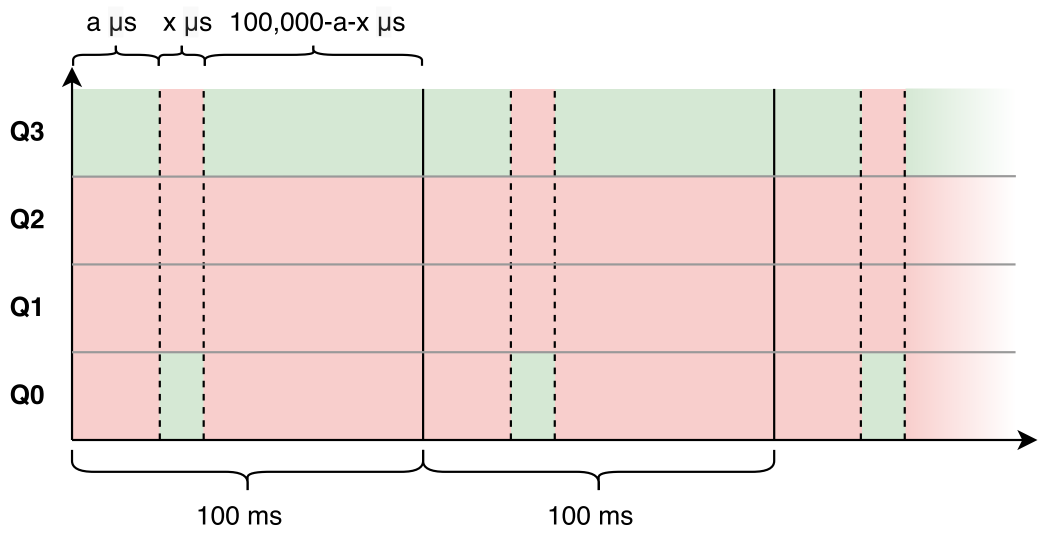


Figure 4.4: TAS gate schedule with 100 ms cycle time. The red and green colors correspond to the gate being closed or open, respectively. The window with size a is a fixed offset of 5 µs, and the window of size x is the window in which a packet will be transmitted.

Queue 0 will be used to test the minimum window size for this test, with Queue 3 being used for PTP synchronisation. Even though only a single end-device is transmitting, the PTP needs to run in order to ensure the clocks between the on-board clock and the clock in the network card are properly synchronised. Even though the PTP service generates packets, the window will be closed while the TSN packets are transmitted, and will thus not cause interference. It should also be noted that the network card checks if there is sufficient time for a packet transmission in a given window, meaning the PTP packets will not have any impact on the test.

To determine if multiple packet transmissions occurs in a single window, the time difference (δt) between packets will be examined. If the delta is given as $0.9 \text{ ms} \leq \Delta t \leq 1.1 \text{ ms}$, then the transmissions are properly separated. Otherwise, two packets are transmitted back-to-back in the same window.

Results

Table 4.2: Statistics regarding the amount of bursts during the tests.

Window size	**Amount of back-to-back transmissions**	**Largest burst of packets**
200 µs	0 (0%)	N/A
190 µs	0 (0%)	N/A
180 µs	0 (0%)	N/A
170 µs	0 (0%)	N/A
160 µs	0 (0%)	N/A
150 µs	2 (0.33%)	2
140 µs	549 (91.5%)	35
130 µs	489 (>81.5%)	63

Table 4.2 shows the results of the tests. It is seen here that the minimum reliable window size with no errors is at 160 µs, with 150 µs being close to reliable. Lower window sizes, however, contain a substantial amount of packet bursts.

When the transmission windows are too small for a reliable transfer, it was expected that the card would queue the frame for the next transmission, i.e. a request to send a frame would be sent in one window, with the actual frame being placed in a buffer instead, and then transmitted during the following window. Instead, large idle periods of approximately 15 seconds occur in which no frames are sent, followed by large bursts of data over a couple seconds. This can be seen in figure 4.5 and 4.6, which shows the frames sent per 10 ms.

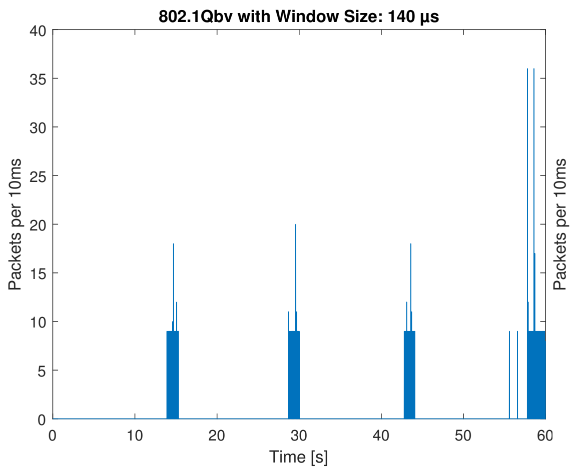


Figure 4.5: Packets transmitted with a window size of 140 µs.

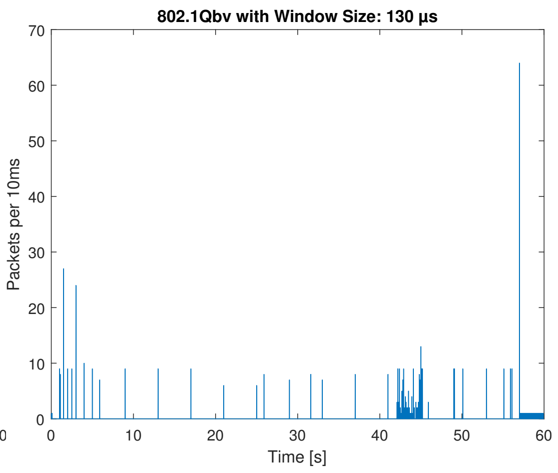


Figure 4.6: Packets transmitted with a window size of 130 µs.

It was for the case of 130 µs observed that approximately 100 packets were dropped by the network card, possibly due to filled buffers. This furthermore lead to the TAS configuration being reset to the general FIFO algorithm, as the buffers may have been overfilled, leading to errors in the card. This lead to the burst of 63 packets within a span of 157 µs, as seen on figure 4.6.

Minimum 802.1Qbv Window Size - Conclusion

It was found that the minimum transmission window, in which a frame of 64B could reliably be sent from the testbench, was of the size 160 µs. Although a a window of 150 µs only resulted in few back-to-back packets, indicating that the actual limit is somewhere between 160 and 150 µs, this is believed to be a sufficient indicator as to when the system will function unreliably. In general, this means that if the window is used exclusively to send a single packet, an idle time of at most $160\mu s - 0.5\mu s = 159.5\mu s$ can be expected.

The given window size is believed to be a result of multiple aspects within the system:

- Communication between the network card and the rest of the system
- Construction of frame
- Buffering into the 802.1Q queues
- TAS protocol
- Check if sufficient time is available to transmit the frame on the interface (1 Gbit/s Ethernet)

However, the time spent from constructing a frame until it is transmitted will in general be much lower, as sending a single frame of 64 bytes per 160 µs will result in a bandwidth of 3.2 Mbit/s. It is therefore suspected that entering a window, i.e. setting the gates on the buffers, takes a significant amount of time. Using exclusive windows for a single frame, may thus have the effect of increasing the overall waste of bandwidth, as switching between windows requires some processing time.

Another issue found with having small transmission windows is the TAS configuration being reset by the network card. This was observed when the window size was set to 130 µs, and is believed to be as a result of the buffers in the network card being filled, resulting in packets being dropped. It should be noted however that the configuration was also found to reset if significantly high amount of packets were being buffered while the queue itself was 'closed'. This test had a low throughput of 10 frames per second, or 51.2 bits/s. The observed burst of 63 frames, or 4032 bytes, might be related to the size of the queues in the network card. It can be seen in the datasheet for the Intel I210 network that it has a 24 KB packet buffer [15]. If part of this packet buffer is shared by the four queues, then it is believed that the entire queue was transmitted when the configuration was reset. This is however not known for certain, and will not be further investigated.

It should be noted that tests with a maximum sized frame has not been conducted due to limitations in *Sample_app_taprio* provided by Intel. This would however give a worst case scenario for the minimum window size which will further show how an increase in payload will influence the window size. It can be seen in [16] that the traffic in an industrial setting consists of frames with small payloads close to the minimum Ethernet frame size, so this result is still relevant in most cases.

4.4 Test Journal - 802.1Qbv Packet Transmissions per Window

Purpose of the Test

Determining the maximum amount of packet that can be transmitted within a given window will give a upper bound on the throughput per window. It was found in section 4.3 that the window size has a lower bound of 160 µs. With this test, it will be possible to determine the the amount of packets that can be transmitted reliably within the minimum window size.

Test Setup

Two computers connected through a single Ethernet cable will be used for this test, as to avoid additional traffic in the network. One computer will be used to transmit UDP packets, while the other will log the received frames using tcpdump.

The TAS will be implemented as described in section 4.2. It will be configured to use the same scheduling as seen for the minimum window size test shown in figure 4.4. The scheduling is further described in section 4.3.

In order to determine if a given amount of frames can be transmitted reliably in the minimum window size, the following procedure will be used:

1. Set the amount of frames to be sent in a window.
2. Run *scheduler.py* followed by *sample-app-taprio.*
3. log data for 1 minute.
4. If the amount of frames is limited by the window, than the maximum is reached.

To quickly find whether the window limits the amount of packets transmitted, a high amount of frames will be set initially. It will then be possible to lower the transmitted packets to the upper bound and keep decreasing until no variations in the throughput occur. Each test will run for 60 seconds, resulting in 600 samples due to the cycle time being 100 ms.

Table 4.3: Number of packets transmitted in a minimum sized window.

Packets pr. Window	Mean	Var	Min	Max
1	1	0	1	1
2	2	0.77	1	4
4	3.99	0.16	1	6
6	6	1.3	0	11
8	7.99	0.43	1	12
10	9.98	0.43	1	12
15	14.98	1.52	1	21
20	19.96	8.22	1	35
25	24.62	54.49	9	72
50	28.14	54	0	62
100	29.32	19.36	1	90
500	30.33	31.54	18	90

Table 4.3 shows the results from tests. One key element in this table is the variance, as it tells how much variation that is seen in the data compared to the mean. It is first seen that the transmission of a single packet per window seem reliable, in terms of the given sample size, as the mean is 1 with no variation. When increasing the number of packets to be transmitted in the window, between 2 and 15, few variations can be found with a mean nearly or exactly the number of packets that should be transmitted. The variance for 8 and 10 packets pr window is the same and is even smaller than for 6 packets, which indicates that these changes in variations is not due to the limitations of the window size. These variations for test between 1 and 15 packets in a window can be seen in figure 4.7.

When further increasing the number of packets transmitted in a window to 20 and beyond, larger variations starts to occur. These variations are very prominent in figure 4.8 when comparing e.g. 20 packet pr. window with 15. As the packets increase from 20, the variations increases as well. When reaching 50 packets and above, it is clear that an upper bound is reached as the mean stays between 28 - 30 packets with some burst reaching 90 packets in a single window.

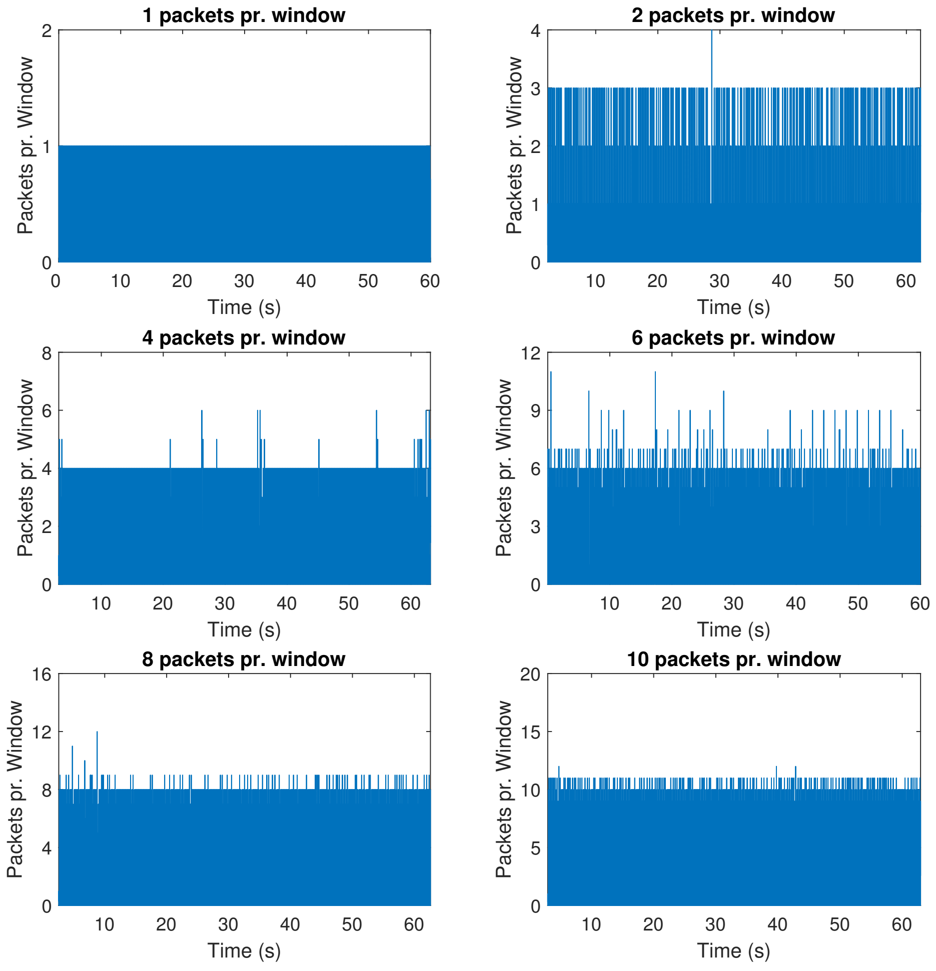


Figure 4.7: Packets transmitted in a minimum sized window of 160 μs.

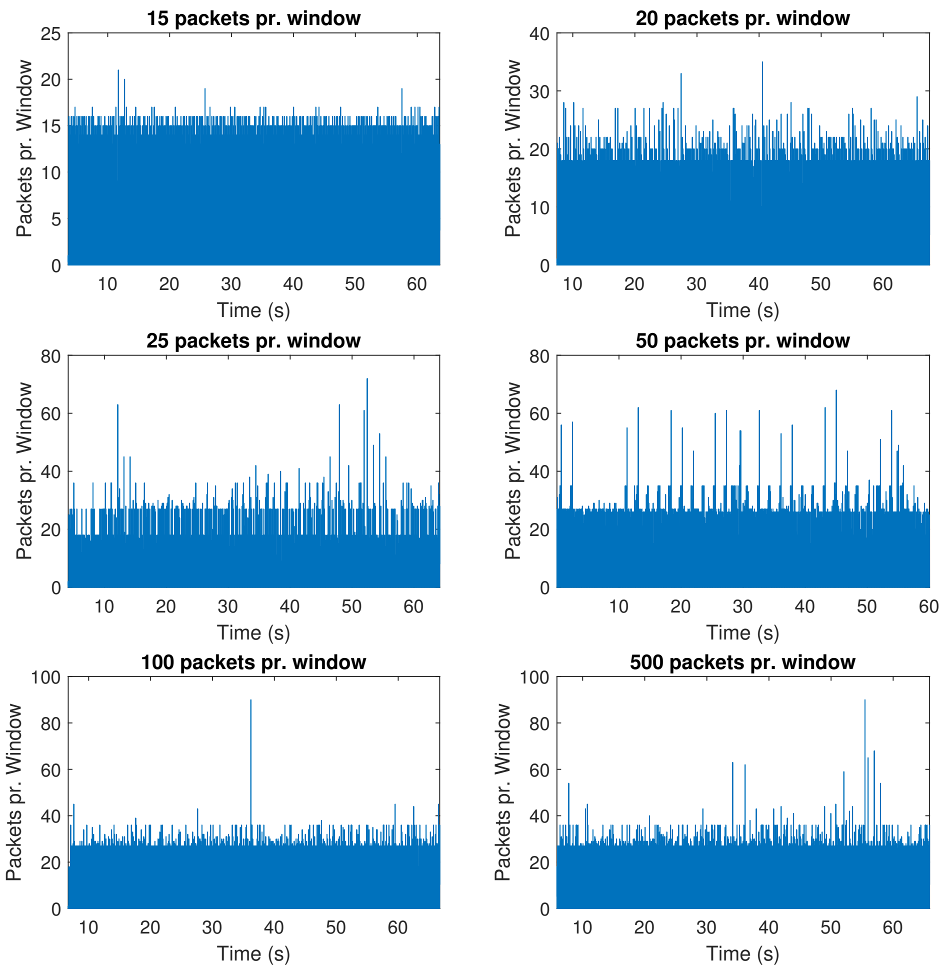


Figure 4.8: Packets transmitted in a minimum sized window of 160 μs.

Packet Transmissions per Window - Conclusion

It was found that the number of packets that can reliably be transmitted within a window size of 160 μs is 1 packet as no variation was found in the given sample size. Although 2-15 packets pr. window could potentially be adequate for some scenarios, when looking at the mean and variance, it is not seen as reliably since fewer packets than intended could be transmitted. This would create problems in TSN scenarios where an expected number of time critical packets are expected to be transmitted at a certain time. It was further observed that the maximum number of packets that could be transmitted in this window was in the range of 28-30. However, the variation in the number of packets transmitted in a single window starts increasing as more than 15 packets is to be sent in a single window.

When looking at figure 4.8 for transmission of more than 15 packets pr. window, large traffic spikes of up to 90 packets are found. These might, as mentioned in the conclusion of section 4.3 be related to size of the queues of the network card which has a 24 KB packet buffer. This is however not known for certain, and will not be further investigated.

4.5 Test Journal - Reconfiguration of Qbv Shaper

Purpose of the Test

It may prove to be beneficial to change the current configuration of a time-aware shaper if the traffic distributions in a network changes. One example of this is adding a new device to a setup while it is running, in which it is then necessary to adapt e.g. window lengths without stopping the entire production. This test aims to determine the impact of reconfiguring the time-aware shaper queueing discipline (qdisc) both while the medium is idle, and when large amount of traffic is present. If all communication is stopped during the reconfiguration, it will be necessary to prepare the devices, as time taken to reconfigure the shaper may exceed their deadlines.

Theory

This test uses the 802.1Qbv time-aware shaper, as described in section 4.1.

Test setup

In order to configure the shaper, the prior configuration, if set, needs to be reset to default. This is done using the command shown in listing 4.6, which targets the interface *enp1s0*.

Listing 4.6: Command to reset the qdisc configuration to default. In this case, the interface *enp1s0* is used.

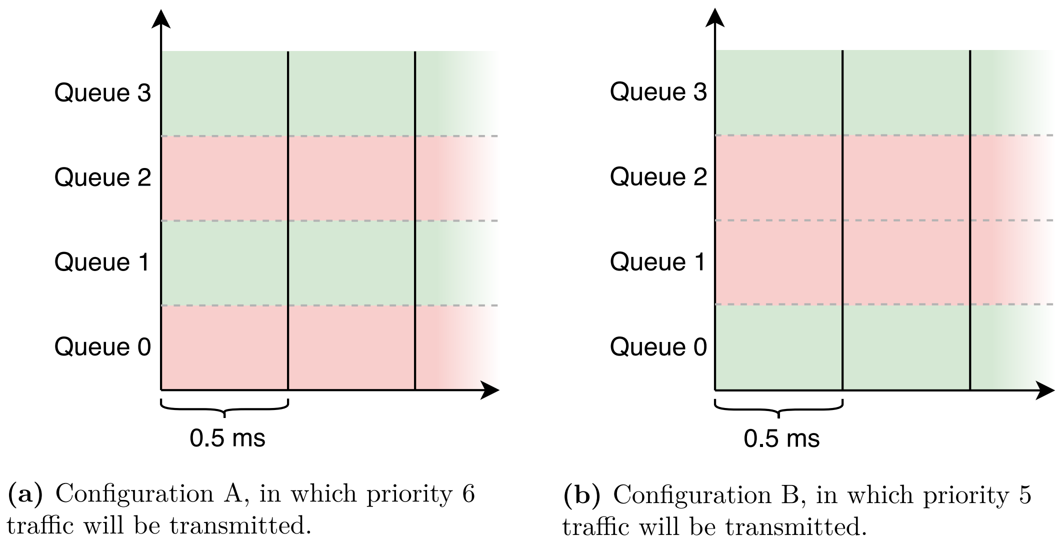
```
$ sudo tc qdisc del dev enp1s0 root
```

Two shaper configurations will be used for this test, shown in figure 4.9a and 4.9b. The shaper will be reconfigured between the two schedules every 2.5 seconds, i.e. from configuration A to B and vica versa. Since the purpose of this test is to determine the effects of the configuration, and not the performance of the scheduling itself, a single window per cycle is used.

The reconfiguration will be tested under three workloads: no traffic, low amount of traffic and near-congestion. For when traffic is present, the gPTP will be used to synchronise the transmitter and receiver, as this is required to properly compare to the receiver using tcpdump. The following data will be transmitted when high traffic is required:

- gPTP packets of priority 7 will be transmitted through queue 3, with an approximate throughput of 1.2 Kbit/s.

- Low traffic
  - 10 Mbit/s traffic of priority 6 will be transmitted through queue 1.
  - 5 Mbit/s traffic of priority 5 will be transmitted through queue 0.
- High traffic
  - 1 Gbit/s traffic of priority 6 will be transmitted through queue 1.
  - 500 Mbit/s traffic of priority 5 will be transmitted through queue 0.



(a) Configuration A, in which priority 6 traffic will be transmitted.

(b) Configuration B, in which priority 5 traffic will be transmitted.

Figure 4.9: TAS configurations used in this test.

The shaper is configured using the command shown in listing 4.7, which uses the configuration A. This command is adapted from that used in the file *scheduler.py*, provided by *iotg_tsn_ref_sw*. The script is described in section 4.2.
To use configuration B, the argument *0xA* is changed to *0x9*.

Listing 4.7: Command to configure the Qbv shaper to configuration A shown in figure 4.9a. In this case, the interface *enp1s0* is used.

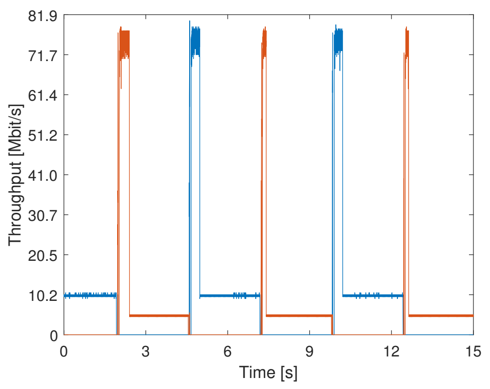
```
$ sudo tc -d qdisc replace dev enp1s0 parent root handle 100 \
taprio num_tc 4 map 2 2 2 2 2 0 1 3 3 3 3 3 3 3 3 3          \
queues 1@0 1@1 1@2 1@3 sched-entry S 0xA 500000             \
clockid CLOCK_TAI
```

Finally, the commands will be run using a script which logs the timestamps of each command.

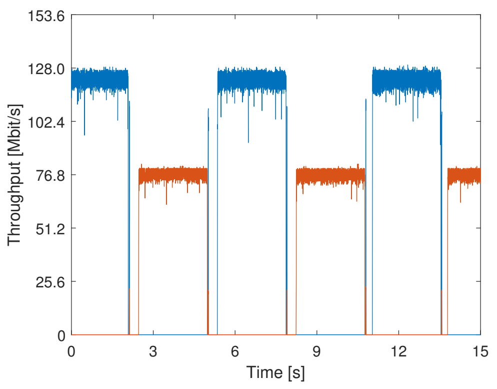
Each test had 50 reconfigurations.

Table 4.4: Statistics regarding the reconfiguration period under various traffic conditions.

Traffic	**Min**	**Avg**	**Max**	σ^2
Idle	98.5 ms	122.6 ms	165.3 ms	0.2
Medium	77.5 ms	106.4 ms	145.1 ms	0.2
High	168.5 ms	330.2 ms	448.9 ms	3.5



(a) Medium traffic. Blue traffic: 10 Mbit/s. Orange traffic: 5 Mbit/s.



(b) High traffic. Blue traffic: 1 Gbit/s. Orange traffic: 500 Mbit/s.

Figure 4.10: Throughput of the network card during reconfiguration of the TAS.

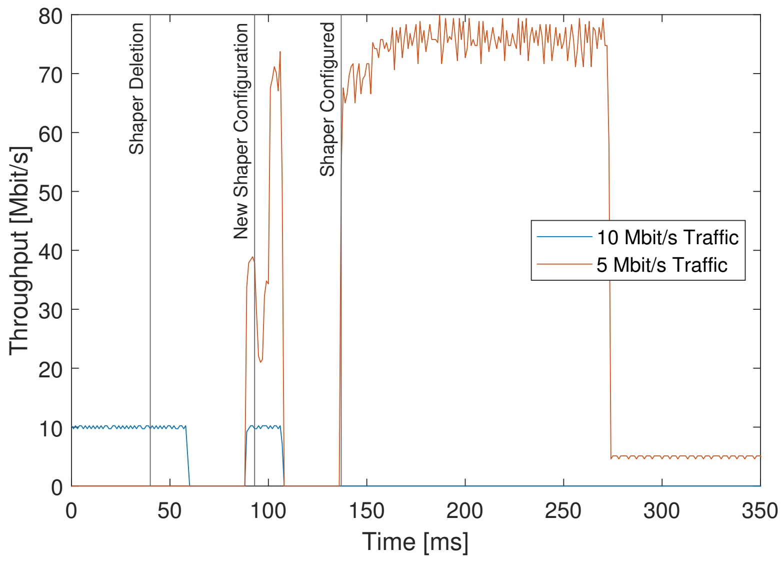


Figure 4.11: Throughput when reconfiguring the TAS with medium load.

Figure 4.11 shows the traffic throughput when the current scheduling is reset to default, and when a new qdisc is being configured. It is here seen that the throughput of all traffic is initially blocked, followed by a relatively large burst from the previously blocked queue (see the orange traffic). When the command for setting the new configuration is sent to the network card, another larger burst of data occurs, followed by another idle period. Finally, when the configuration is set, a burst of high throughput for 130 ms is sent, whereafter the throughput is lowered to that of the transmitter, i.e. the buffers are empty. This also means that the size of this burst will depend directly on the amount of data being generated.

Several observations can be found from this test:

- Both when deleting and setting a configuration, the output of the network card will be blocked entirely. This can cause congestion of the medium when the output is opened again.
- There are several instances of bursts from the queue which was previously blocked.
  - The initial burst indicates that the shaper is disabled, allowing all frames to be transmitted in a FIFO manner.
  - The second, larger burst is after the shaper has been set. The amount of data being transmitted indicates that the buffer is being emptied, after which the data is transmitted at the same rate it is being generated from the application.
- The amount of traffic has an impact on the reconfiguration.

- When idle or with a medium amount of traffic is present, the reconfiguration period is of similar lengths. Although the transition is seemingly faster when there is traffic, this is believed to be due to the low sample size. Furthermore, while the bandwidth used by the bursts are significantly higher than that generated by the applications, they do not use the entire bandwidth available.
- When very high amounts of traffic is present, the reconfiguration period is significantly longer. The numbers from table 4.4 do not take into account the following burst of data, which however cannot be seen when the interface is already being congested.

Reconfiguration of Qbv Shaper - Conclusion

Changing the 802.1Qbv shaper configuration during transmissions lead to brief periods of downtime and bursts of data during the reconfiguration period. The reconfiguration was found to require 77 to 165 ms both when no traffic was being sent, and with 15 Mbit/s traffic. For a congested connection (1 Gbit/s), this period increases to 168-448 ms. These downtimes can be unbenificial for time-sensitive traffic since these can cause the QoS requirements of devices to not be fulfilled. If a configuration change for the TAS is required, it will be necessary to prepare the other systems, as critical errors may otherwise occur due to the large latency or packet loss.

Chapter 5

Installation of Omnet++

Following the guide from https://doc.omnetpp.org/omnetpp/InstallGuide.pdf, this is the step by step installation process of Omnet++ or Omnetpp in Ubuntu 18.04.1. In the guide, you may have to jump in chapters to make the install, therefore here is the continuous sequence:

Before starting the installation, refresh the database of available packages. Type in the terminal:

```
$ sudo apt-get update
```

There are some prerequisite packages listed in chap 5.3 of the guide. To install the required packages, type in the terminal:

```
$ sudo apt-get install build-essential gcc g++ \
  bison flex perl \python python3 qt5-default \
  libqt5opengl5-dev tcl-dev tk-dev \libxml2-dev \
  zlib1g-dev default-jre doxygen graphviz \
  libwebkitgtk-1.0 openscenegraph-plugin-osgearth \
  libosgearth-dev openmpi-bin libopenmpi-dev \
  libpcap-dev nemiver
```

Download the omnetpp file from the omnet website https://omnetpp.org/download/ and open a new terminal:

```
$ cd Downloads
$ tar xvfz omnetpp-5.5.1-src-linux.tgz
$ cd omnetpp-5.5.1
$ . setenv
```

(Note that the command *setenv* have down space between . and setenv.)
The *setenv* command gives the path for the enviroment, which needs to copied, e.g. /home/user/Downloads/omnetpp-5.5.1.
In the terminal, run the command:

```
$ nano ~/.bashrc
```

and add the following line at the end of the file, then save it:

```
export PATH=\$PATH:/home/user/Downloads/omnetpp-5.5.1/bin
```

Here, user is your pc-name. Close and re-open the terminal for the changes to take effect. Omnet++ can now be built and configured:

```
$ cd Downloads
$ cd omnetpp-5.5.1
$ ./configure
$ make
$ make install-menu-item
$ make install-desktop-icon
```

Omnet++ can now be launched using the shortcut on the desktop or using the terminal in the folder:

```
$ omnetpp
```

Reconfiguring the Libraries

```
$ cd omnetpp-5.5.1
$ ./configure
$ make cleanall
$ make
```

5.1 Framework - CoRE4inet

The framework CoRE4inet is used, and are located at: `https://github.com/CoRE-RG/CoRE4INET`. It works with OMNeT++ 5.5.1 and INET framework 3.6.6.
By installing Omnet++ 5.5.1, the guide for installation will work and the application will launch, where you choose a folder as workspace, which will not contain inet and the examples.

1. Get INET framework 3.6.6
   - `https://inet.omnetpp.org/Download.html`
   - move into workspace
   - File -> import.. -> existing Projects into Workspace -> select inet folder
2. Install CoRE plugins
   - OMNEST/OMNeT++ -> Help -> Install New Software...

- URL: http://sim.core-rg.de/updates/
- Check/Enable:
  - Abstract Network Description Language
  - CoRE Simulation Model Installer
  - Gantt Chart Timing Analyzer

3. Get CoRE framework
   - CoRE Simulation Model Installer:
   - OMNEST/OMNeT++ -> Help -> Install CoRE Simulation Models...
4. Working with the framework
5. Installation successful

5.2 Simulation - CoRE4inet

In Omnet++ there some different language needed. They all use the programming language C++, where the essentials is: basics, data types, control structures, functions, object oriented and advanced concepts. In OMnet++, there are .INI files, .NED files and xml files. The .INI and .NED files use basics, data types, control strcutures and functions. The xml have syntax, elements, attributes, XPath, NED functions and c++ functions.

In the framework CoRE4inet, there are several network examples in different aspect of TSN.

1. avb_AS6802
2. IEEE8021Q
3. IEEE8021Qbv
4. IEEE8021Qci
5. inet
6. tsn

Chapter 6

Omnet++ simulation - 802.1Qbv

This chapter contains information regarding modules and how they are built of components, in which various parameters are defined. Furthermore, the configuration and setup of a test networking using the 802.1Qbv Time-Aware Shaper (TAS). The descriptions and tests will be using the CoRE4inet framework, presented in chapter 5 section 5.2.

6.1 Decomposition of Modules and Components

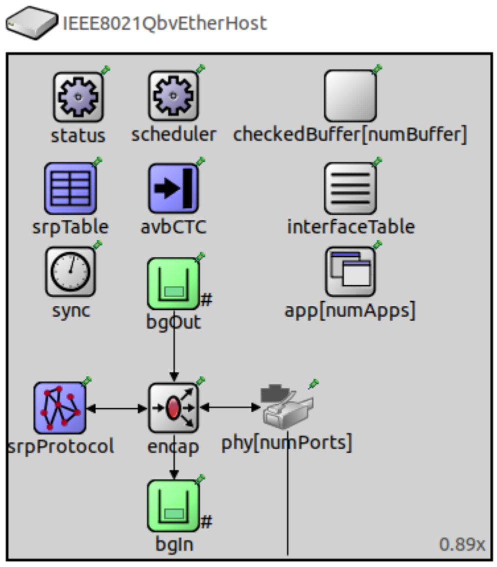


Figure 6.1: The host module with its components. This can be compared to an end-device in a physical setup.

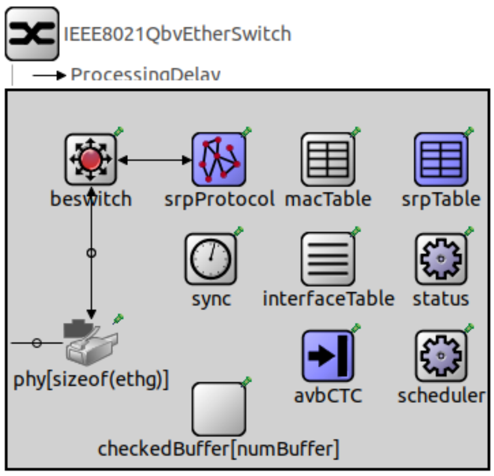


Figure 6.2: The switch module with its components.

6.2 Small network - 802.1Qbv

The aim of this section is to demonstrate a small IEEE 802.1Qbv Network. The Traffic generators send Ethernet frames directly. The small network used for the tests consists three hosts (node1, node2, node3) and a switch, which connects these hosts as illustrated in figure 6.3.

Node 1 sends Q traffic in vlan 0 with pcp 7. The TAS gates in the devices are configured to handle it like TDMA traffic. Node 2 sends AVB Class B Stream 1 in vlan 2 with pcp 6. The switch uses a CBS to shape this traffic. Cross traffic may be sent via all nodes.

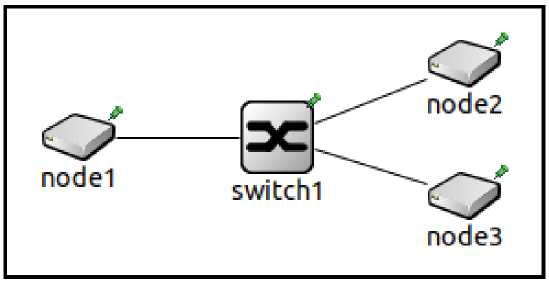


Figure 6.3: The setup of an test network

The nodes (1 to 3) are the EtherHost and switch1 is the EtherSwitch, as defined in the section 6.1. Each node and switch needs to be configured plus the same as the overall network.

The overall network has a defined the tick length to be 80 ns, while 1 period has a cycle of 500 µs. The network also has clock synchronisation, with the maximum clock drift at 200 ppm and a maximum clock drift change of 0.1 ps per cycle. The precision of the synchronisation is 500 ns.

In the switch, some of the things there has to be configured: the Stream Reservation (SR) class is set to B and the gate control is as follows::

$$\begin{gathered}
\text{"}o,o,o,o,o,o,o,C : 0; \\
C,C,C,C,C,C,C,C : 0.000125; \\
C,C,C,C,C,C,C,o : 0.00025; \\
o,o,o,o,o,o,o,C : 0.000375\text{"}
\end{gathered}$$

The gate control syntax in each line is a window, where first is the eight priorities (0-7) which is open (o) or close (C). The activation time in the period defined in ms after the

':'. Note that the cycle ticks are defined in this example to 500 ns = 0.0005 s.

Finally, the nodes has to be configured. In the nodes the application (talkers/listeners) are defined. Figure 6.4 shows the traffic of figure 6.3.

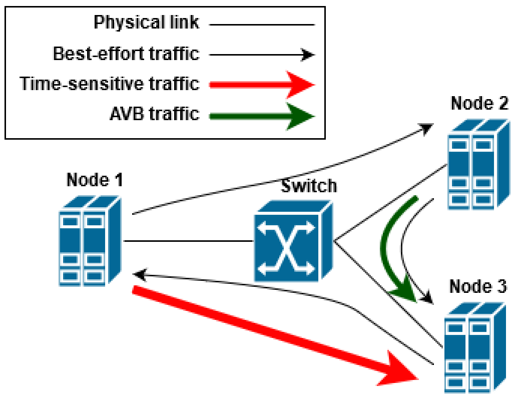


Figure 6.4: Traffic of the small network

For each arrow (traffic), there has to be an application in either end (talker and listener).

Node 1 contains two talkers and one listener. The talker is Time-Sensitive, and is named *app*[1, 1], where the numbers define the node and application, respectively. This talker has priority 7, payload of 1500 Byte and sends a packet each 500 us, the same as every cycle. The corresponding gate control:

$$\begin{aligned}&"C, C, C, C, C, C, C, C : 0;\\ &\;C, C, C, C, C, C, C, o : 0.00015;\\ &\;o, C, C, C, C, C, C, C : 0.0003"\end{aligned}$$

The second talker *app*[1, 2] is background traffic, best-effort. This talker also has a payload of 1500 Byte and sends with a interval of 500 us. The last application of node 1, *app*[1, 3] is a listener.

Node 2 also contains three applications; two talkers and one listener. The first talker *app*[2, 1] is the green arrow which is AVB Class B stream in vlan 2 with pcp 6. The payload of the packets has the size 750 Byte, with the following gate control:

$$"o, o, o, o, o, o, o, o : 0"$$

The second talker *app*[2, 2] is identical to *app*[1, 2]. The last application *app*[2, 3] is a listener.

Node 3 have four applications, three listeners and one talker. The talker $app[3, 1]$ is similar to the previous best-effort applications, $app[1, 2]$ and $app[2, 2]$. Then the $app[3, 2]$, $app[3, 3]$ and $app[3, 4]$ are listeners.

6.3 Test Journal - 802.1Qbv Packet Transmissions per Window

Purpose of the Test

Before the simulation tool can be used to determine the performance of various networks, it is necessary to ensure that it provides similar results to that of the testbench. The two will therefore be compared when using identical TAS configurations.

Theory

The shaper used in this test, the 802.1Qbv, is presented in the worksheet 4, and will not be explained in this worksheet.

In order to compare the performance of the two implementations, the frame latency will be utilised. Since the processing time by the TSN-enabled network card is of interest, the latency will be measured as follows:

- For the simulation, this is measured from when a frame is sent from the transmitter-node, until it has travelled through the switch and arrived at the intended receiver-node.
- For the testbench, this is measured from when a frame is generated and sent to the network card for transmission, until it arrives at the receiver.

If the latencies are of similar magnitude, i.e. if the latencies from the simulation are not instant, then the processing time in switches may have been adapted to a realistic setting. This is beneficial, as the results will be closer to that of a physical setup.

Test Setup

Both the simulation and the testbench will utilise the TAS schedule shown in Figure 6.5. This schedule has been chosen to primarily allow for Strict Priority Scheduling to have the most significant effect, while also blocking some traffic sources momentarily.

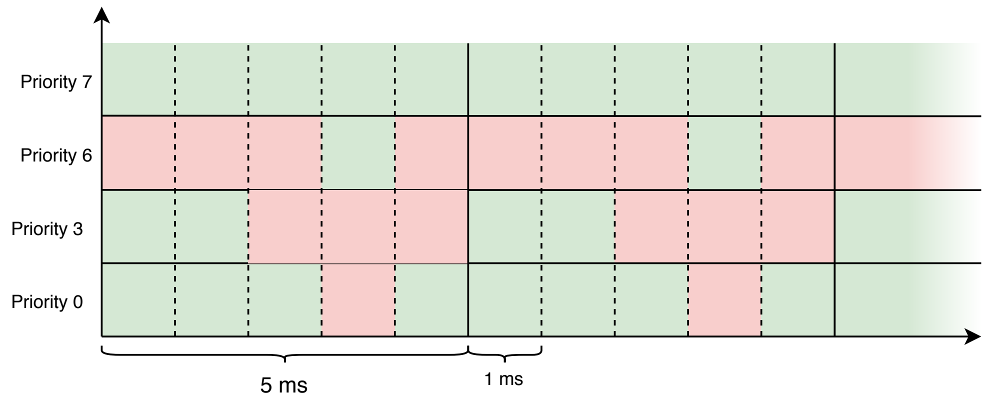


Figure 6.5: TAS schedule used for simulation and testbench.

The traffic is generated as shown in Table 6.1. Priority 7 will be reserved for the gPTP in the testbench, as this is required to determine the frame latency. A single second of

Table 6.1: Traffic sources used in simulation and on the testbench.

Traffic type	Priority 6	Priority 3	Priority 0 / Best-effort
Send interval	0.1 ms	0.025 ms	0.01 ms
Payload size	18 Bytes	18 Bytes	18 Bytes
Bandwidth	5.12 Mbit/s	20.48 Mbit/s	5.12 Mbit/s
Expected frames/s	10,000	40,000	10,000

traffic will be used to determine the latencies, corresponding to 200 cycles of the TAS schedule.

Simulation

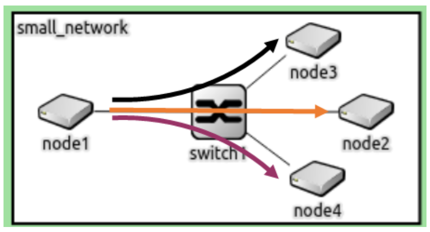


Figure 6.6: Test setup of network in Omnet++. The black arrow indicates the traffic flow for priority 5, the orange arrow for best-effort traffic, and the pruple arrow for priority 6.

Figure 6.6 shows the network used for this test in OMNET++ using the CoRE4inet framework, where node 1 contains the three traffic sources as described in table 6.1.

Nodes 2, 3 and 4 each represent a listener. The TAS is configured in the switch, which works similar to that of the Intel I210 network card.

Other configurations for the network include the synchronisation and gate control. The data from the gPTP test is used to emulate its accuracy in an attempt to further emulate a realistic setting. The synchronisation is configured as follows:

- Synchronisation precision: 16 ns
- Maximum clock drift: 2800 ppm
- Drift change: Uniform distribution between −75ppm and 75ppm.

The TAS schedule is configured as described previously. The gate control in this framework consist of a binary value (o for open, C for closed) for each priority within a defined window period. The value after ':' defines the window period.

$$\begin{gathered} o, C, C,\ \ o, C, C, C, o : 0 \\ o, C, C, C, C, C, C, o : 2 \\ C, C, C, C, C, C, o, o : 3 \\ o, C, C, C, C, C, C, o : 4 \end{gathered}$$

The end-to-end latency can be obtained by inspecting the scalar-values provided by the framework.

Testbench
The TAS on the testbench is configured using the tools provided by IotG [8], more specifically the *scheduler.py* script. The exact procedure to apply the TAS is shown in worksheet 4.2.

Three different configurations of the *sample-app-taprio* traffic source will be used, with each configuration being directed to its own queue. Although the sources are configured to follow the specified TAS schedule, they will instead be set to periodically transmit a single packet following Table 6.1. This will result in frames being queued, similar to the simulation.

In order to determine the latency, the same script as the traffic sources is used, as the *sample-app-taprio* program also supports functionality for receiving frames. The receiver can then by inspecting the payload extract a timestamp just before frame was transferred to the network card, and subtract it to the time the frame was received. It is configured to do so using the command shown in listing 6.1.

Listing 6.1: Command to obtain frame latencies using *sample-app-taprio*.

```
$ cd sample-app-taprio/
$ sudo ./sample-app-taprio -i enp1s0 -x 2 -q "7 3 0" -y 1
```

Here, *-x 2* sets the program to receiver mode only, *-q "7 3 0"* specifies that only frames with these priorities should be inspected, and *-y 1* outputs the data to the terminal. This data can then be saved and further inspected to obtain the desired statistics.

Results

Because of difficulties implementing the traffic on priority 6 in the simulation, this has been changed to use priority 7 instead. This will result in this traffic source taking precedence over all other sources. This will not have a negative impact on the results, as the corresponding change is being made on the testbench. While this results in the gPTP traffic being mixed with other frames in the same queue, it has been found to remain reliable in terms of synchronisation, as shown in worksheet 2.3.

Table 6.2: Frame Latency in TAS Configuration. The data is obtained from 1 second of traffic, with each transmitted frame consisting of 64 Bytes.

Traffic	Min [ms]	Mean [ms]	Max [ms]	σ^2	Frame Count
Experimental Results					
Best Effort	0.0073	0.1370	1.0556	7.2211e-8	99834
Low Priority	0.0082	0.9567	3.0349	1.0377e-6	10000
High Priority	0.0078	0.0124	0.5732	1.6501e-10	37566
Simulation Results					
Best Effort	0.0092	0.1157	1.0037	5.9425e-8	100,000
Low Priority	0.0105	1.6522	4.0029	1.7234e-6	9,990
High Priority	0.0092	0.0095	0.0105	1.1492e-13	40,000

Conclusion

It is clear from the results that the simulation provides similar latencies to the physical testbench, with the largest difference i the mean latency being approximately 7 µs. This indicates that the simulation tool may be used for larger TAS networks, improving testing conditions both in terms of hardware cost and time spent on implementation.

Chapter 7

Per Stream Filtering Policing

The IEEE 802.1Qci standard, also known as Per Stream Filtering Policing (PSFP), is an amendment to the 802.1Q, introduced to improve the robustness of the demanding industrial network. These complex industrial networks made it necessary to introduce 802.1Qci features to mitigate on the vulnerability of a complex network and hence deliver frames more reliably to their destination.
The 802.1Qci focuses mainly on traffic being received at the ingress port. Its purpose is to mitigate the effects of switches that operate incorrectly by transmitting messages at arbitrary points in time, thus corrupting the transmissions on the network [17]. It does this by defining the “filtering and policing functions" which include the detection and mitigation of disruptive transmissions by other systems in a network and improving the robustness of that network [18].

7.1 Stream filtering

The following paragraphs on the functionality of the features within 802.1Qci is based on the description given in the TSN amendment [18].

Per stream filtering is intended to provide administrative control over the use of the shared medium and therefore the limitation of the unwanted traffic by dropping the unwanted MAC addresses. The stream filter also increases the overall throughput by eliminating unnecessary traffic on the network. It furthermore reduces the load placed on the end devices caused by the reception of the unwanted frames.

When frames arrive on the ingress port they are treated as follows and is illustrated in figure 7.1;

- Data which is persistent within a given network (e.g. data loops) and MAC addresses not appearing in MAC table are prevented in the Forwarding Process by a defined value of TRUE or FALSE. The learning and forwarding control mechanism depends on the received frame and the switch port. If learning is "true" for both the reception port and the ingress, filtering cannot cause the received frame to be dropped. The source MAC address and the VID are submitted to the Learning Process. The Learning process contains a MAC Address Table for each switch.

The Active topology enforcement is therefore used to minimise the denial of service following any change in the physical topology of the network. This can be implemented by creating a redundant network and applying Spanning Tree Protocol (STP) mechanism. When STP is implemented on a redundant network, any link which is faulty and causes data loops in the network will be blocked and traffic rerouted through the healthy routes, and hence prevents bandwidth starvation.

- The Ingress Filtering parameter may be supported on each ingress port. This makes it possible to discard any frames whose VLAN ID has not been learned. The Ingress filter is disabled by default for all ports and can be configured administratively.

- When the frames are to be forwarded to the egress port, the forwarding mechanism makes the filtering decision for each received frame. The decision is based of the information entered in the MAC address table and by looking at the default group filtering behaviour [19].

- The Forwarding Process queues each received frame to each of the transmission ports that can process frames with VID.

- The Forwarding Process may then apply flow classification and metering to the frames on the outbound switch port. The flow classification identifies a subset of frames, that may be subject to the same QoS such as rate-limiting implementations in terms of metering and forwarding. Whereas the flow classification rules may be based on destination MAC address, source MAC address, VID and priority [19]. The flow meter on the other hand can change the drop-eligible parameter associated with each frame. It can also discard the frames on the basis of the parameters for each frame and the previously received frames, and the time elapsed since since the frames were received. These parameters are the received value of the drop-eligible parameter and the mac-service-data size. The flow metering is always applied after any other filter specifications as illustrated in figure 7.1 below.

- The Forwarding Process provides storage for queued frames waiting to be transmitted according a predefined algorithm such as 802.1Qav and 802.1Qbv described in chapter 3 and 4, respectively.

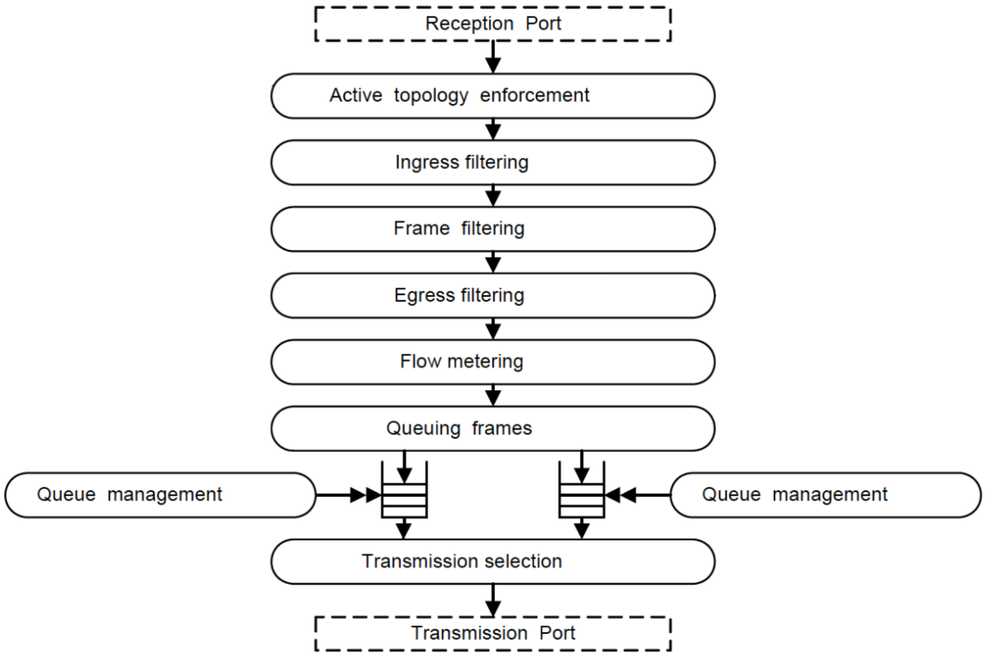


Figure 7.1: A Forwarding process function [18, Modified].

The stream filter uses rule matching to allow frames with specified stream ID's, priority levels and apply policy actions as required. These policy actions are described in the stream filter instance table consisting of an ordered list of stream filters. Each stream filter contains some of the following highlighted elements:

- The stream filter identifier (stream ID) contains an integer value which is used to uniquely identify the filter instance and is used as an index on the table. The list of streams are entered in the table with the smaller identifier values appearing earlier on the ordered list.
- The priority specification, which can be single priority value, or a wild-card value that sets the priority to zero.
- The stream gate instance, which identifies the stream gate instance used by the stream filter. The gate can either be open, letting frames pass through, or closed, in which frames are blocked.

Note that the PSFP implementation can be done on the layer 2 devices like switches and routers. From the TSN perspective, the 802.1Qci operates at the switch and at end station devices, and is limited to the devices that support it. This is an area worth doing more research and tests on when it comes to the implementation of the dynamicity of the TSN end devices. The section can be used as reference and to see how the TSN standard is built.

Chapter 8

Packet Preemption

When transmitting packets of different priorities, it is necessary to ensure that both the prioritisation is being upheld, as well as that no packets are dropped because of these prioritisations (assuming normal traffic). In order to migrate these issues, there are several methods applicable, of which three will be presented in this worksheet: Guardband, 'Look-Ahead' and Preemption.

The first method is to insert a guard band, i.e. an idle-period, in between time slots which has the size of a maximum frame size used by system to ensure that even in a worst case scenario, the frame will be fully transmitted before a higher priority packet can be sent. This method will however delay the transmission of the higher priority packet. An illustration of this guard band can be seen in figure 8.1.

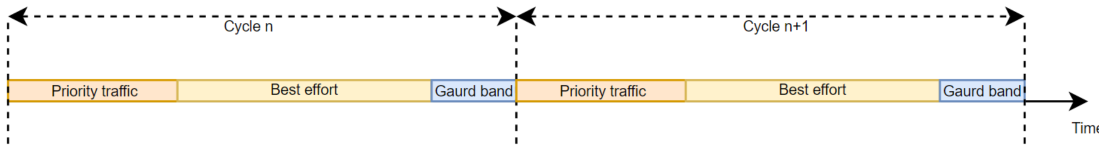


Figure 8.1: Illustration of how guard bands could be implemented. Best effort traffic (yellow) is seen as the lowest priority and priority traffic (orange) is non-best effort [14].

The second method is the one used in the 802.1Qbv implementation on the testbench, which describes a Time Aware Shaper (TAS). The TAS controls the time controlled traffic gates, which are presented in chapter 4. With this method, guard bands are not necessary to prevent frame interference. Instead, the remaining time is checked to ensure the entire frame can be transmitted. One drawback of this method however is not fully utilising the entire transmission gate open time, which can be seen as undesirable in terms of resource utilisation [8]. An illustration of this method can be seen in figure 8.2.

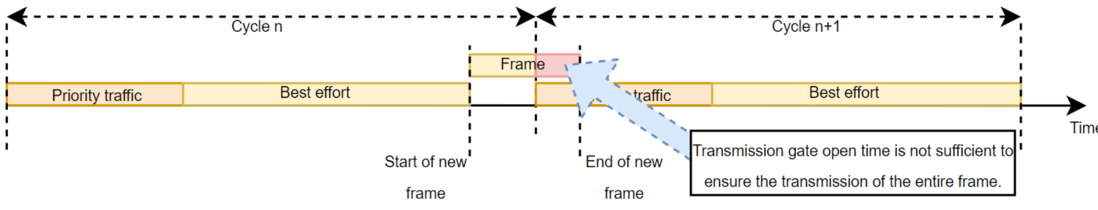


Figure 8.2: An illustration of how the transmission gate open time in this case is not sufficient to ensure the transmission of the entire frame. The frame is therefore not sent until the next applicable window. Best effort (yellow) is seen as the lowest priority and priority traffic (orange) is non-best effort. [14]

The third option is introduced in 802.1Qbu, where preemption is used to further improve the efficiency of bandwidth in a network. Preemption is a method which allows higher priority frames to interrupt the transmission of a lower priority frame and thereby break lower priority frame into fragments. When an interruption of the frame in progress occurs, a 4-byte checksum is transmitted to the receiver as to indicate that the frame transmission is not complete [20]. Once the higher priority frame has been transmitted, the lower priority transmission can then be resumed. This is seen in figure 8.3 with a time division implementation, as for 802.1Qbv, where a best effort frame is interrupted due to packets of higher priority. As soon as the prioritised traffic in complete, the remains of the interrupted frames will be transmitted [21]. This method utilises the entire transmission gate open time, if implemented with 802.1Qbv and will thereby mitigate the lost transmission time seen with the guard band and the 'Look-Ahead' algorithm. This method will not be looked into further for this project, but is nonetheless an important feature for improving the utilisation of the bandwidth on a time-sensitive network.

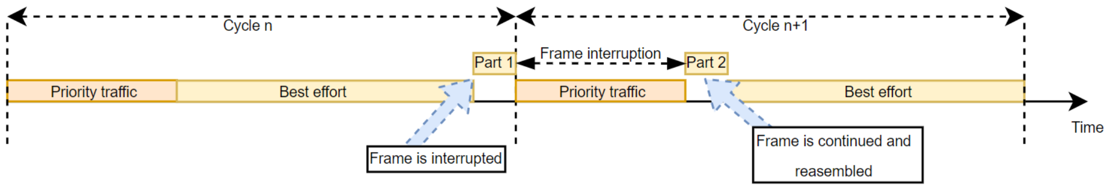


Figure 8.3: An illustration of preemption. Best effort (yellow) is seen as the lowest priority and priority traffic (orange) is non-best effort. [14]

Chapter 9

TSN Notes

This worksheet should be seen as general notes for understanding TSN, and takes starting point in `https://www.youtube.com/watch?v=kOOM4LAUqVY`.

Common model based on internet protocols and the IEEE 802 architecture, which have operation of **best-effort** (quickest) but not suitable for use in cases that require high/-known availability like circuit switching networks or fieldbusses for control networks.

Notice: operation network (OT) is not the same as information network (IT).

Therefore, Time-sensitive Networking (TSN) is a set of evolving IEEE standards that aims to address resource availability, so that bandwidth reservation and bounded latency are provided in Ethernet-based LANs. TSN standards are becoming widely adopted with different applications on several domains - automotive infotainment and control, industrial control automation (fieldbus), professional A/V, etc - with interoperability playing a key role on the ecosystem development.

GOAL: m ake Ethernet able to handle time sensitive traffic in addition to best-effort traffic to background traffic to everything.

Effect: TSN allows for both time-sensitive traffic and best-effort traffic to coexist on the same network. The whole point of TSN is making sure the time-sensitive traffic is deterministic, in other words have to provide a bounded worst-case latency.

TSN started from audio/video bridging group (**AVB**), which allows for OT and IT traffic to co-exist. TSN provides bounded worst-case latency, which is a requirement when the Ethernet run on deterministic traffic.

If OT and IT traffic can co-exist on the same network, the amount of cabling can be reduced, the weight of the product is also reduced which can positively affect other areas.

TSN is being expanded to the standard **802.1Q**, virtual LANs (VLANs) and QoS. Most of the amendments to **802.1Q** is e.g. the **802.1AS**, which is the mechanism for time synchronisation. Operation of TSN consists of the time syncchronisation, traffic identification (**VLAN tags**), resources allocation and traffic shaping/scheduling. The traffic shaping consists of most of the amendments to **802.1Q**.

One amendment is **802.1Qcc**, which is network configuration for TSN. In the network configuration, the network can be set to be dynamic/static and distributed/centralized. One example of such **802.1Qcc** is the **SRP** (Stream Reservation Protocol).

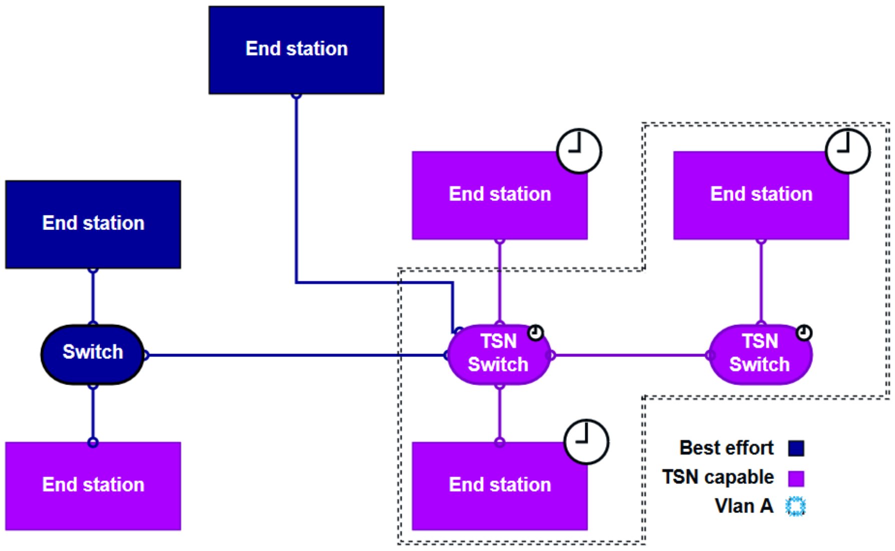


Figure 9.1: An example of a network with cross-traffic.

Figure 9.1 shows an example of a network with cross-traffic, consisting of TSN and non-TSN traffic with a common network card. There are end-stations with TSN capable devices connected to the non-TSN network because everything can coexist. To the right of the figure, the TSN domain is found on the network. In the TSN domain, all the clocks are synchronized; each end device and switch has the same time domain.

Because time-sensitive traffic and best effort traffic both must be able to coexist on the network. At this point one traffic has a higher priority than the other traffic, but somehow the network has to be able to identify such traffic correctly.

VLAN is used to identify the traffic in the network. The network will then know how to prioritise one traffic over the other. It is also necessary to allocate resources as well in the network, so that all devices (end stations, switches, listeners) agrees on how to allocate the resources along the network. Using traffic shaping to ensure the TSN traffic behaving well all together of these mechanisms is quite complicated. Somehow it is needed to be able to configure the network. The standard that is trying to put together different ways to configure the network for TSN is 802.1Qcc.

TSN: traffic shapers

TSN applications have two different requirements; reserved bandwidth and strict cycles (scheduled). Traffic shapers are defined in some of the amendments: **802.1Qav** is the Credit-Based Shaper (CBS), which is a per-queue bounded bandwidth and "transmit all packets from this traffic class at X kbps". **802.1Qbv** is the Time-Aware Shaper (TAS) and is a per-packet Tx time, and said to transmit this packet at timestamp 152034537600000000 ns.

is it "not earlier than" or "not later than"?

The **802.1Qbv** is enhancements to scheduled traffic, use per-port queues schedule and said "execute the Tx algorithm on queue 0 every 100us for 20us, on 1 every 240us for 30us.

other Shapers is the **802.1Qbu**, **802.1Qci** etc...

Traffic shapers are basically **bandwidth management**, a way to distribute traffic evenly in time. In different application domains, there are different requirements. For example, if all you have for the time-sensitive traffic is reserved bandwidth; fine. But for some other systems that has higher deterministic requirements; strict cycles are needed because the transmission of packets are going to be scheduled.

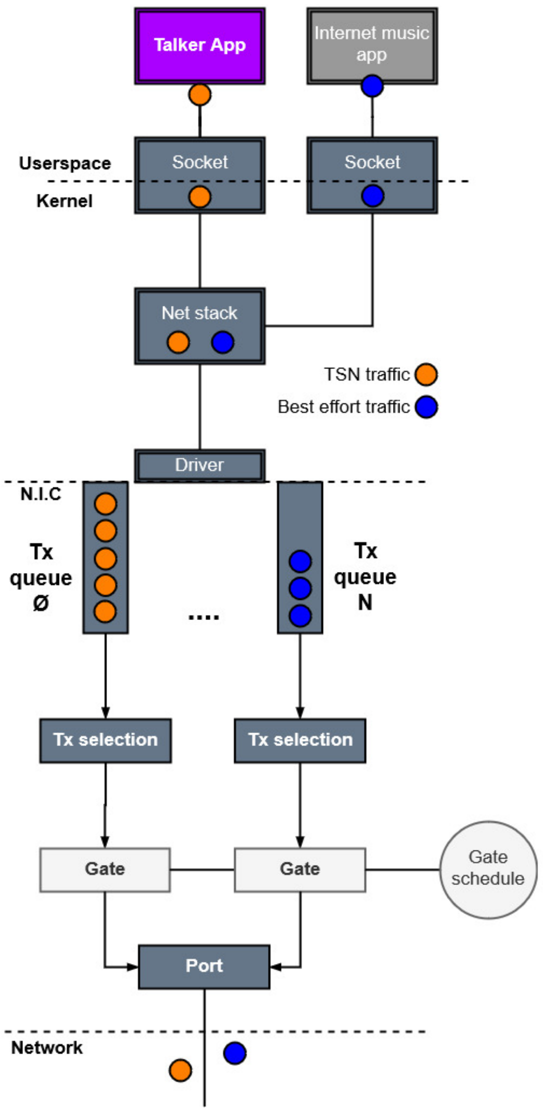
Qav := bounded bandwidth, so if all that is needed is that for a given traffic class, the bandwidth never go beyond certain threshold.

If more fine-grained control of the packets over the transmission time is required, the time scheduling shaper can be applied. If you have a per packet transmission time, then the shaper controls when that packet will hit its deadline.

Qbv := per port schedule; a full schedule of every single transmission queue on the system. **Qbu** := frame pre-emption **Qci** := not even a shaper, it reserves the stream.

TSN on End stations Primer

The **talker** application has enabled multi queue, which need to be configured using a shaper. The **talker** application has classified traffic to steer to Tx queue and allow the network to identify it and the **talker** application transmit. The **listener** application have optionally Rx filters setup, i.e. VLAN priority, src and dst MAC. The **listener** receives the data.



This figure shows the end station **talker**, which has both best effort traffic and TSN traffic. A talker application and an internet streaming app - best effort. The traffic go to the network stack and then to the NIC. A requirement for NIC that are TSN capable, that they have at least two transmission queues. One is used for time-sensitive traffic and the other is for best effort. The more the better. One each one of these queues, there will be a transmission algorithm running. The algorithm is actually a shaper. If the NIC is compliant with **Qbv**, there may be a gate schedule running here that allows the gates to run at certain points.

Current upstream TSN SW architecture

TSN SW began with previous attempts.

1. OpenAVB
   - First Eric Mann's from Intel worked on the AVB, where the project were named **OpenAVB**. The OpenAVB bypasses kernel network stack, have a forked driver: **igb__avb** and the configure and data paths: **libigb**. Tx Queues exposed directly to the userspace.
2. RFCs
   - Henrik Austad from CISCO worked on making RFCs on netdev, which is Media centric (AVB), bundled up as a TSN driver. a **ConfigFS** based interface. **ALSA** shim for audio streaming.
3. Driver-specific interfaces on upstream
   - The main is **Stmmac** and maybe others: device tree as a config interface for shapers. downsides: kernel bypassing, hw-dependent, monolithic solutions.

The **OpenAVB** is a demo, a driver:**igb__avb**. bypassed the kernel and exposed al the transmission queues and the registers to the user space through a library -> **Open Avenue. AVNU Alliance**.

Henrik Austad work -> stream times as a traffic over the network. Made two interactions on the work, but the maintainer didn't like it because it was bundled up.

Found out there were a few drivers upstream that are exposing the shapers configurations through device tree. So this is all very hardware-specific.

Downsides of these previous work: they were all hardware dependent or doing kernel bypassing or there were very few monolithic.

Traffic control on Linux

There was no stream support for TSN, but Linux has a traffic control subsystem. The traffic subsystem already provide interfaces for shaping/scheduling (Tx) and policy (Rx) etc.. the components from the traffic control subsystem on Linux are basically achieving disciplines (qdiscs, classes and filters).

Qdiscs = packet buffers inside the kernel, so they live between the protocol families and device drivers. They are **kernel buffer** for packets. The kernel buffer sits "between" protocol families and net device driver. Control when/how packets are transmitted.

Every network interface has a qdisc attached to it, at least one (root qdisc). They can expose inner classes in which can install children qdisc. Qdiscs can offroad work to hardware.

Mqprio qdisc: enable multi queue priority, it "exposes" HW queue as classes, allowing for other inter qdisc to be attached. It maps priorities to traffic classes to HW queues.

$$\text{\$ tc -g qdisc show dev } \textbf{INTERFACE} \tag{9.1}$$
$$\text{\$ tc - g class show dev } \textbf{INTERFACE} \tag{9.2}$$

```
$ tc qdisc replace dev INTERFACE \
parent root mqprio num\_tc 3 map 2 2 1 0 2 2 2 2 (...) \
queues 1@0 1@1 2@2 hw 0
```

This is a configuration using **mqprio**, where there are 3 traffic classes and 4 transmission queues.

- Prio 3 -> tc 0 -> queue 0 (8001:1)
- Prio 2 -> tc 1 -> queue 1 (8001:2)
- Other -> tc 2 -> queues 2(8001:3) and 3 (8001:4)

CBS

The credit-based shaping (802.1Qav) were there developed the **cbs qdisc**. This feature were available from kernel 4.15, debuted with **Intel i210** support only, but more to follow. The CBS provides both HW offloading and SW fallback. OBS! CBS is bandwidth-centric.

Example: configure CBS for traffic class 1 (priority 2)

```
$ tc qdisc replace dev INTERFACE parent 8001:2 \
cbs locredit −1470 hicredit 30 \
sendslope −980000 idleslope 20000 offload 1
```

$ tc -g qdisc show dev **INTERFACE** (9.3)

It is directly from the standard: need the same low credit, the high credit, same slope and idle slope. the idle slope is the bandwidth that your traffic class requires.

Now installing the CBS qdisc on to the traffic class one. next to dump the classes, to see that the CBS is installed.

TBS

The Time-based scheduling, there are developed the **tbs qdisc** and the **SO_TXTIME** socket option. It was developing with Richard Cochran (linuxptp maintainer). It provides both HW offloading and SW fallback. It can be found at http://patchwork.ozlabs.org/cover/882342/ . This were also debuted with i210 support only.

tbs qdisc can hold packets until their TxTime minus a configurable delta factor. it sort packets based on their TxTime, which is optional and only before they are sent to the device queue. tbs is time-centric and therefore need a peer-packet timestamp.

Example: configure TBS for traffic class 0 (priority 3)

```
$ tc qdisc replace dev INTERFACE parent 8001:1 \
tbs clockid CLOCK_REALTIME delta 150000 sorting offroad
```

$ tc -g qdisc show dev **INTERFACE** (9.4)

First installing the TBS qdisc on the traffic layer 0 and setting a delta parameter of 150 microseconds. Which means, if the first packet is supposed to be transmitted on two seconds from now, then the qdisc is gonna hold that packet until (2 sek - 150 micro seconds) then dequeue the packet into the net device.

example; 10 applications on the user space and they're all sending traffic on the same traffic class, where have different periods. the network card gonna block packets. the head of the queue is going to block all the packets behind it. if the packets are out of order -> result in a packet in the future back there is supposed to be sent in 2 sec from now blocking a packet that's supposed to be sending on one second from now. if it hits the net device before. the qdisc sort of the packets instead of the user space. So if turn on, the qdisc can sort the packets based on their transmission time meanwhile it's holding packets inside its buffer. The qdisc doesnt work by itself, it needs a per packet timestamp.

Data path: socket interface

Regular sockets is used to transmitting data. To TBS a new socket option **SO_TXTIME** is used for enabling the feature for a given socket. a **cmsg** header is used for sting a per-packet **txtime**, an a **drop_if_late** flag. reference clockid_t will become a socket option argument.

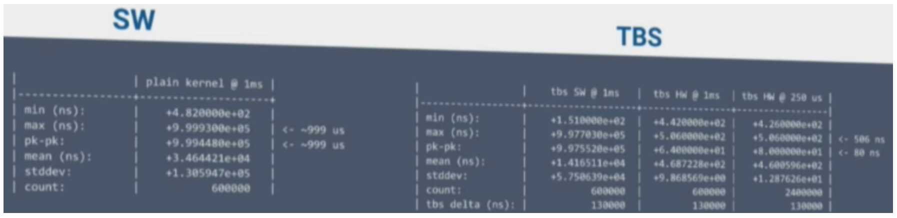
tx_time is a new socket option, which basically enables the feature and let the kernel know that for that socket there are valid timestamps on the packets, it needs to be copied into the socket buffers. addition the clock id (per packet clock id) is actually a socket option argument. can set the transmission time, a flag (drop_if_late). The question:

- no later than
- not earlier than

TBS is not a standard shaper, it's not on an IEEE document. TBS is basically common sense. depending on the application domain, it maybe a strict deadline or not, or a soft deadline. -> therefore this flag to tell of the packets should be dropped if delayed.

The preferred method of classifying traffic is the socket option **SO_PRIORITY**, where it flag all packets with a specific priority. but **iptables or net_prio cgroup** can also be used. the priority is later used as the PCP field of the VLAN tag of the ethernet header. Steers all traffic from the socket into the correct HW Tx queue. OBS! have to setup a mapping for that with the **mqprio qdisc**.

Results - TxTime based Scheduling



```
|                   | plain kernel @ 1ms |
|-------------------+--------------------+
| min (ns):         |      +4.820000e+02 |
| max (ns):         |      +9.999300e+05 | <- ~999 us
| pk-pk:            |      +9.994480e+05 | <- ~999 us
| mean (ns):        |      +3.464421e+04 |
| stddev:           |      +1.305947e+05 |
| count:            |             600000 |
```

```
|                   |    tbs SW @ 1ms   |   tbs HW @ 1ms   | tbs HW @ 250 us |
|-------------------+-------------------+------------------+-----------------|
| min (ns):         |     +1.510000e+02 |    +4.420000e+02 |   +4.260000e+02 |
| max (ns):         |     +9.977030e+05 |    +5.060000e+02 |   +5.060000e+02 | <- 506 ns
| pk-pk:            |     +9.975520e+05 |    +6.400000e+01 |   +8.000000e+01 | <- 80 ns
| mean (ns):        |     +1.416511e+04 |    +4.687228e+02 |   +4.600596e+02 |
| stddev:           |     +5.750639e+04 |    +9.868569e+00 |   +1.287626e+01 |
| count:            |            600000 |           600000 |         2400000 |
| tbs delta (ns):   |            130000 |           130000 |          130000 |
```

this picture is from a test, looking on only SW and TBS. The test had following setting:

1. DUT: i5-7600 CPU @3.50GHz, kernel 4.16.0-rc2+ with about 50usec maximum latency under cyclic test
2. ptp4l + phc2sys
3. packet size: 322 bytes all headers included

There is min, max, pk-pk, mean, stddev, count.. All is standard for statistics, beside of pk-pk which is peak to peak aka jitter.Can do time based transmission using nothing but software, application can ex. call clock and use asleep. This test look at the different result using pure SW and TBS. This test is designed by Richard Cochran, when the first version of TBS, now this test is used as a baseline.

This example is run on the machine mension in the three bullet, which is a fine machine. in the test, sending packet every 1 ms and the transmission period is 200 microseconds.TSN dont care about average, only the worst case because of deterministic.
SW: worst case is almost 1 ms (999 us), in other word almost a whole entire deadline and it is bad to miss a whole deadline. even with a good machine.
TBS: better in everything compare to **SW**; min, average, max etc..Here it have enable all; the hardware offload, performing sorting etc.. Here the worst case is at 506 nanoseconds, jitter = 89 nanoseconds.
OBS!! need to configure the qdisc, maybe the delta factor need to be slightly different(here around 120 microseconds). using a high resolution timer inside the qdisc.

User Space

AVNU Alliance member maintain the **OpenAVNU** evolution of OenAVB. it provide with daemons, libs, examples and frameworks.

- GPTPd: 802.1AS
- MrPd: SRP daemon

OpenAVNU is mostly focused on the Pro A/V domain. where it recent contribution from intel: **libavtp**. **libavtp** provides packetization for applications that use AVTP as a transport (`https://github.com/AVnu/OpenAvnu`). Part of the user space is **linuxptp**.

- ptp4l: precision time protocol implementation for linux
- phc2sys: synchronizes the PTP Hardware Clock to the System Clock

Discuss the challenges ahead

Next challenge is to configure the interfaces; 802.1Qbv and 802.1Qbu. The 802.1Qbv is the enhancements to scheduled traffic and the 802.1Qbu is frame preemption. One idea for a new qdisc-based interface: **taprio (time aware priority)**

taprio is time-aware version of mqprio, a part of CBS RFC v1: `https://patchwork.ozlabs.org/cover/808504/` the push-back: there were no NICs for end stations with support for these standards. There are required to provide a SW fallback, therefore may re-consider an ethtool based interface instead. TBS could be used, but that requires a scheduler for converting the per-port schedule from Qbv into a per-packet txtime.

Looking at the **data path**, the linux network stack is very good for throughput, it's designed for data centers. TSN (specially for industrial control) require not only bounded latency, but bounded low latency. maybe the linux network stack is cable of it. eXpress Data Path looking at that problem.

Wrap up

- TSN aims to provide bounded latency on Ethernet based LANs
- SW interfaces for Linux are starting to become available upstream starting with the cbs and tbs qdiscs
- future work aims to address other traffic shapers (802.1Qbv / Qbu)
- low latency is (probably) an issue. There are efforts trying to reduce the bounded worst-case latency of the Linux network stack: AF_XDP
- userspace building blocks are also gaining traction
- openAVNU is becoming the consolidator of TSN SW components for userspace
- Zephyr will have TSN support soon!

Bibliography

[1] I. I. Consortium, "Time sensitive networks for flexible manufacturing testbed characterization and mapping of converged traffic types," [Online] Available from: https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Char_Mapping_of_Converged_Traffic_Types_Whitepaper_20180328.pdf, 03 2019, retrieved: 05/11/2019.

[2] IEEE, "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*, pp. 1–292, March 2011.

[3] E. Technologies, "Precision time protocol (ptpv1)," *https://endruntechnologies.com/pdf/PTP-1588v1.pdf*, UNKOWN, read: October 16 2019.

[4] D. Arnold, "What makes a master the best?" *https://blog.meinbergglobal.com/2013/11/14/makes-master-best/*, 2013, read: October 16 2019.

[5] J. L. University of New Hampshire, InterOperability Laboratory, "Ptp background and overview," *https://www.iol.unh.edu/sites/default/files/knowledgebase/1588/ptp_overview.pdf*, 2012, read: November 04 2019.

[6] P. D. Henning Puttnies and more, "A simulation model of ieee 802.1as gptp for clock synchronization in omnet++," *https://www.microsemi.com/company/technology/ieee-1588-technology#how-1588-works*, 2018, read: October 16 2019.

[7] AVnu, "Github repository for OpenAvnu," https://github.com/AVnu/OpenAvnu, 2018.

[8] Intel, "Github repository for Intel reference software," https://github.com/intel/iotg_tsn_ref_sw, 2019.

[9] HP, "Qos - managing bandwidth effectively," *https://techhub.hpe.com/eginfolib/networking/docs/switches/RA/15-18/5998-8155_ra-2620_atmg/content/ch04.html*, 2018, read: November 1 2019.

[10] IEEE, "Ieee standard for local and metropolitan area networks–bridges and bridged networks," *IEEE Std 802.1Q-2014 (Revision of IEEE Std 802.1Q-2011)*, pp. 1–1832, Dec 2014.

[11] IEEE, "Ieee standard for local and metropolitan area networks - virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams," *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. C1–72, Jan 2010.

[12] ——, "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-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, March 2016.

[13] P. P. LUXI ZHAO and S. S. CRACIUNAS, "Worst-case latency analysis for ieee 802.1qbv time sensitive networks using network calculus," *https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8418352*, 2018, read: November 1 2019.

[14] Wikipedia, "Ieee 802.1qav forwarding and queuing enhancements for time-sensitive streams," *https://en.wikipedia.org/wiki/Time-Sensitive_Networking*, 2019-28-03, read: September 5 2019.

[15] Intel, "Intel Ethernet Controller I210 Datasheet," https://www.intel.com/content/www/us/en/embedded/products/networking/i210-ethernet-controller-datasheet.html, 2019.

[16] I. Rodriguez, R. S. Mogensen, E. J. Khatib, G. Berardinelli, P. Mogensen, O. Madsen, and C. Møller, "On the Design of a Wireless MES Solution for the Factories of the Future," in *2019 Global IoT Summit (GIoTS)*, June 2019, pp. 1–6.

[17] C. Temple, "Avoiding the babbling-idiot failure in a time-triggered communication system," in *Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (Cat. No.98CB36224)*, June 1998, pp. 218–227.

[18] IEEE, "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)*, pp. 1–65, Sep. 2017.

[19] I. C. Society, "Ieee standard for local and metropolitan area networks— media access control (mac) bridges and virtual bridged local area networks," *https://www.standards.ieee.org/standard/802_1Q-2012.html*, 2012, read: October 4 2019.

[20] I. E. Book, "Preemption standard enables high priority frames and traffic," *https://iebmedia.com/index.php?id=11934&parentid=63&themeid=255&hft=97&showdetail=true&bb=1*, 2017, read October 23 2019.

[21] IEEE, "Bridges and bridged networks — amendment 26: Frame preemption," *https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7553415*, 2014, read October 22 2019.