Simulating Time-Sensitive Networking

Harri Laine

Kongens Lyngby 2015

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

The goal of the thesis is to ...

Preface

This thesis was prepared at DTU Compute in fulfilment of the requirements for acquiring an M.Sc. in Engineering.

The thesis deals with ...

The thesis consists of ...

Lyngby, 26-June-2015

Harri Laine

Acknowledgements

I would like to thank my....

Contents

Chapter 1

Introduction

The number of different types of devices connected to the Internet and other networks is increasing at a fast pace [1]. These new types of devices can unleash new opportunities that have not even been thought to be possible. However, in some places the requirement to transmit time-critical data is not fulfilled by Ethernet, meaning the Internet, Local Area Networks (LAN) and Metropolitan Area Networks (MAN) available today [24]. Different use cases require different properties from the network and real-time systems have more requirements. Time-criticality is usually used in real-time systems where it is not only important to get the data, but to get the data at the right time [14].

When data is transferred for example to backup files over LAN or to access LAN services, such as printers or e-mail, there is no need for predictability. It is, of course, convenient to get the wanted data quickly, but lag or jitter is not an important issue as it does not make the system act abnormally. It only becomes problematic when it slows down working so much that the system becomes unusable. It is a throughput issue rather than misbehavior issue and therefore it can be fixed easily by increasing bandwidth in the network.

Multiplayer games on the other hand, are affected by lag. Players are interconnected and their actions can be dependent on each others' actions. For example, let us consider that two players are connected to the same server and player 1 has no lag and player 2 has 1 second of lag. Now if player 2 constructs a building

in spot X, the information is on the server only one second later. During that one second player 1 could do something that interacts with spot X, which would result the game to behave incorrectly for one of the two users depending on the algorithm to solve conflicts. Hence, low lag gets more important than just high throughput. The lag is more important if the game is more fast paced or there are multiple players as the conflict comes more likely.

Requirements are even higher, when video and audio are streamed with high quality. This can be the case for example in mobile television transmission trucks, which relay television stream for example from sports event to regional distribution centers. High quality video and audio transmission uses much more bandwidth than network gaming and these are also sensitive to problems in the connection, such as jitter or packet loss.

In some places, it is can be life-threatening to receive the data later than its deadline. These systems need predictability from the network and are called real-time systems. Some real-time systems are safety-critical (hard real-time) and some are not (soft real-time). For hard real-time systems the data has to be received at the right time. Failure to get the data at the right time might result to application misbehavior or even death. For example, if brakes do not work in a car because the data is not received in time, the results can be catastrophic. If the system is soft real-time, the result of delayed data will be seen as loss of quality. This can be seen in streamed video when video frames will be skipped or have some errors. On the other hand, the same network may have a need for copying a file using LAN. That is best effort (BE) data and the requirement is high throughput rather than predictability.

LANs have usually been used to connect computers to the Internet or within a company network to connect computers to each other and to servers. Therefore there has been a need for separate network to meet the requirements for deterministic communication (see section 1.2) in factories and automotive industry. Determinism can be defined as "a physical system behaves deterministically if, given an initial state at instant t and a set of future timed inputs, then the future states and the values and times of future outputs are entailed" [14]. That means the network's behavior can be predicted. For deterministic network needs there are different networks, such as FlexRay, Controller Area Network (CAN) and Time-Triggered Protocol (TTP) that meet the requirements [29][16].

Having these components with different needs and requirements in same system results in mixed-criticality system. It can be defined as being "an integrated suite of hardware, operating system and middleware services and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure compute platform" [3].

1.1 Internet of Things

When devices, appliances and other elements, such as transport and security, is connected to the Internet, it is called Internet of Things (IoT) [1][19]. Some of the devices are more critical than others. For example locks and other elements that are related to security are should take priority over streaming music. Interconnectivity between all elements, from computers to fridges, is playing a major role in IoT making the communication protocol important part of IoT [19]. For that reason, it is reasonable to extend already available networking possibilities instead of creating new ones.

1.2 Need for deterministic networking

Nowadays, for example in a factory, the machinery can be controlled and supervised from a distant location. Depending on the situation, the ability to control machinery can have the requirements for safety-critical networking. Different standards and laws define the required safety level for certain applications. For example, a car's software safety integrity level is defined by how uncontrollable the vehicle will be if the given software fails [23].

In a factory environment, there are normal PCs for employees to do their work and there can be machinery controlled by computers. Employees have a need to access LAN for internet and intranet services. Machinery uses network to communicate with the computer controlling it. The needs for connected machinery and normal PCs can differ a lot. Machinery can be safety-critical, while employees need a normal LAN connection and care mostly about throughput. ref

These use-cases need better solutions than the ones generally available today. Reducing costs can be achieved by having only one network, which meets the requirements for deterministic networking and is compatible with present day solutions.

1.2.1 How deterministic networking works

Deterministic networks are predictable. In order to be predictable, the data goes through the network as expected every time. For example if two replicated components have same initial state, they will finish at about the same [14]. In order to accomplish this in a network, there cannot be traffic blocking time-critical

data. Some bandwidth must be reserved in order to meet the requirements. have to do some reservation along the communication path. It means zero congestion loss and guaranteed latency [8]. Because of the guarantees and the fact deterministic network is predictable, if there is a lost packet, it counts as failed hardware. Deterministic networks can be used to connect hard real-time systems.

1.2.2 Reliability matters

Probably everybody has some experience of voice-over-IP (VoIP) or streaming audio or video. Streaming services show lost or late packets very clearly by producing problems in the audio or video. With deterministic networking, these problems will not happen as the network can guarantee certain bandwidth and delays for data resulting in better quality for end users. However, the true benefits for deterministic networking become evident for example in industrial or automotive use, where even one packet being lost or late may cause catastrophic consequences. Hence, deterministic networks are primarily developed for industrial needs [12].

1.3 Different networks

The following sections describe differences in network designs and where to use them. Some implementation examples are also described. Because there can be some proprietary network protocols with various features, only the fundamental differences between time-triggered and event-triggered protocols are explained.

1.3.1 Time-triggered vs. event-triggered

There are many different protocols for data transmission in a network. Some of them are pure event-triggered, some are pure time-triggered and some are mixed. Event-triggered protocols do not provide any timing guarantees, but overheads stay relatively low as data can be transmitted as soon as it arrives.

Time-triggered protocols know when communication should happen, which makes it useful for time-critical systems. Downside for preallocated timing is that if some data is ready for transmission just after a cycle has started, it will have to wait for the next cycle to be transmitted [15]. This makes time-triggered

protocols potentially having lower throughput than event-triggered. Differences between purely time-triggered and event-triggered data transmission is shown in figure 1.1 where the vertical arrows from a to d define the points when data is ready to be submitted and darker gray part of the horizontal arrow is when the data is actually transmitted..

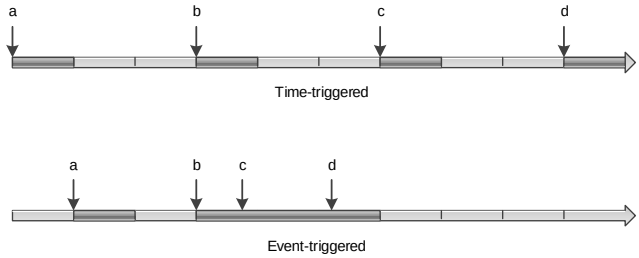


Figure 1.1: Time-triggered vs. event-triggered protocol (redo fig).

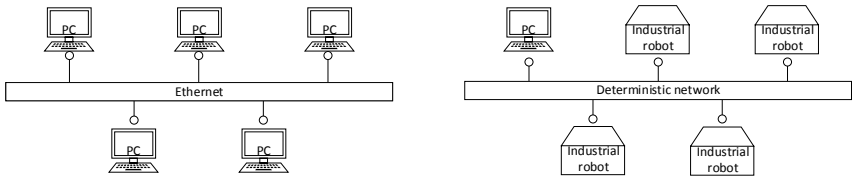
The figure shows that time-triggered data arrives or is sent at predefined times and it can be guaranteed to be schedulable. Pure time-triggered protocols can have extra capacity, which is hard to take utilize fully. On the other hand, event-triggered protocol can have too many things to transmit at the same time and therefore not be able to meet timing requirements.

1.3.2 Network architectures

When building a network or designing a system using network, the network must be chosen according to requirements for it. However, many times there is a need for Ethernet network to be available as it is used to connect computers. Other components, such as machinery in factory, may have stricter requirements and that means there is a need for two networks. As shown in the figure 1.2a, the network contains only PC's and is homogeneous network. The network in figure 1.2b is a control network, where one computer controls four industrial robots. The network has to be deterministic for example for safety reasons.

It is simpler and cheaper to have one network only and therefore there is a need for heterogeneous networks. The network has to have a protocol that meets the requirements for all the devices shown in figure 1.3.

Even though the example network has only one device type, industrial robot, that requires a deterministic network, the whole network has to support it. There are multiple protocols providing proprietary deterministic networking [29]. However,



(a) Ethernet network **(b)** Deterministic network

Figure 1.2: The two different networks needed in a factory environment

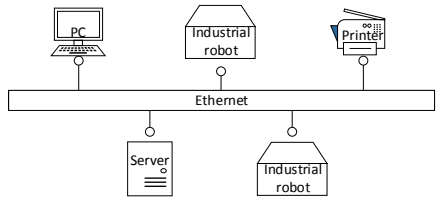


Figure 1.3: Heterogeneous network

there are not many protocols that are compatible with widely used protocols, such as Ethernet. One of the few examples of Ethernet compatible protocols is TTEthernet, which is able to handle both time-triggered and event-triggered data with speeds at 100 Mbps and above [27].

1.3.3 Network protocols

Costs can be reduced by reducing the number of different networks needed. Using generally available hardware and protocols instead of proprietary ones is can also reduce costs as there can be many suppliers pushing the price down.

TTEthernet can cut down the costs of having different networks – it supports mixed-criticality with Ethernet. It can be used in heterogeneous networks (such as figure 1.3) due to its compatibility with Ethernet [27]. TTEthernet contains three different types of data: time-triggered (TT), rate-constrained (RC) and best-effort (BE). These three data types have different requirements and therefore are handled differently. The protocol has different integration methods determining how resources are used between higher priority and lower priority data. These features make TTEthernet usable in places where cost, efficiency, safety and predictability are a great concern [29].

Another protocol for mixed-criticality systems using Ethernet is Time-Sensitive

Networking (TSN). TSN is a set of standards on top of IEEE 802.1Q and its amendments, which defines most importantly Virtual LANs (VLAN) and prioritization. While IEEE 802.1.Q has support for Quality of Service (QoS), it was not designed to be used in industrial or automotive networking, but was meant for better service quality for audio and video [9][28]. TSN addresses this issue and is reviewed more thoroughly in chapter 2. IEEE 802.1Q has some advantages and some disadvantages compared to TTEthernet, but TSN is supposed to target the disadvantages [12][28][26].

1.4 Problems in data transmission

There can be many different problems in data transmission. These include, for example, packet loss, lag, jitter and congestion. These problems for time-critical data are main concerns in TSN. One of the biggest problems appears when data is flowing through many input ports to one output port or from faster connection to a slower connection. When some of the data has to wait for its turn to be transmitted it increases latency. When there are many nodes to flow through and wait for transmission, the latency may grow too large.

1.4.1 Congestion and packet loss

When too much data is being transmitted in a network, it is said to be congested. Congestion can result in packet losses, because the network cannot handle all the data. Lost packets have to be retransmitted in some cases and sometimes those have to be ignored at the expense of degraded quality of service.

For example, transferring a file has to result in a perfect copy and therefore retransmission is needed, but for a live video stream the lost packet can be ignored if it belongs to a frame that has been already shown in the viewer's receiver.

1.4.2 Latency and jitter

Latency is the time observed from the beginning of something until it is done. In networking it is he time it takes for packet to be delivered from the source to the receiver is called latency. It can be used also in round-trip delay times.

Jitter is a definition for latency differences between selected measurements. In other words, it measures the variation of latency and can be used to determine stability for a network connection quality. Jitter can be reduced up to a certain point without end user noticing extra delays [7]. This can be achieved with buffers adding delays, thus decreasing latency differences.

1.4.3 Converged networks

While optimally congestion, packet loss, latency and jitter should be negligible, they still exist and in hard real time applications the effects can be catastrophic. The network has to be able to adapt and handle different kinds of data as well as possibly. The need to have exact copy of a file, streaming or controlling industrial equipment is converged in future networks.

1.5 Objectives

The main objectives for this thesis is to create a simulator and simulate IEEE 802.1Qbv, IEEE 802.1Qav and IEEE 802.1Qbu. These different features will be also used in combination in series of test sets. The objectives are:

- Model both action and event-oriented simulation paradigms
- Model different IEEE 802.1 features for network
    1. Qbv (TSN time-aware shaper)
    2. Qav (AVB credit-based shaper)
    3. Qbu (preemption)
- Implement a simulator with the three different features
- Determine average and worst-case latencies for non-time-critical messages
- Determine jitter for non-time-critical messages
- Evaluate results and compare with features' design goals

1.6 Structure

n finished

This thesis has the following structure:

- Chapter 1 - Introduction
- Chapter 2 - Time-Sensitive Networking
- Chapter 3 - Modeling and simulation
- Chapter 4 - Simulator design and implementation
- Chapter 5 - Evaluation
- Chapter 6 - Conclusion

Chapter 2

Time-Sensitive Networking

This chapter discusses the need background for TSN, technical details its architecture. An example is shown for making it easier to understand how it works in practice.

When different types of data has different priorities, the data packets have to be distinguished from each other. Each node in a network must know how important it is for a certain packet to get transmitted. The standard IEEE 802.1Q and its amendment IEEE 802.1p were introduced for this purpose. These standards make it possible to tag data frames and define priority levels.

Usually the amount of time-critical data is low, when compared to other types of data [12]. It is therefore possible to prioritize transmission in a way that allows time-critical data to be transmitted when required without blocking all the other data. This should result in be predictability and high throughput.

Research on the standard was started by IEEE 802.1 AV Bridging Task Group (AVB), which aimed to "provide the specifications that will allow time-synchronized low latency streaming services through 802 networks", so the use of standard Ethernet or WLAN is possible [9]. This satisfies to need for cutting costs and using generally available hardware, such as cabling.

2.1 Background

The IEEE 802 standard is split into different standard sets for different areas in networking. TSN belongs in IEEE 802.1, which is a group for bridged networks and network management. IEEE 802.1's latest amendments are the research area for TSN.

2.1.1 Standards used in TSN

Different features are used to make the network perform better in different kind of situations. Standards in 802.1 are concerned with how to interconnect networks and data priorities, for example.

Table 2.1: IEEE standards explained or used in this thesis

IEEE standard[a]	Features/area of interest	Notes
802.1	Bridging and network management	
802.1Q	Virtual LANs	
802.1p	Traffic class expedition and priorities	[b]
802.1AS*	Timing and Synchronization	
802.1ASbt*	Timing and Synchronization: Enhancements and Performance Improvements	
802.1Qbu*	Frame Preemption	
802.1Qbv*	Enhancements for Scheduled Traffic	[c]
802.1CB*	Frame Replication and Elimination for Reliability	
802.1Qcc*	Stream Reservation Protocol (SRP) Enhancements and Performance Improvements	

[a] Starred standards used in TSN development.
[b] Merged into 802.1D, and later was merged into 802.1Q-2014 [10].
[c] This thesis' main focus.

The table 2.1 shows standards which are important for TSN. 802.1Q with its amendments is the basis for prioritizing data and tagging it. The following section describes data tagging in more depth.

2.1.2 Data frame tagging

These standards allow data to be tagged adding a 16-bit header to a Ethernet frame [6]. The tag consists of 12-bits Virtual LAN (VLAN) ID, 3-bits for priority

and 1-bit for drop eligible indicator [6].

Other fields in the 802.1Q header are priority for each data frame, priority code point (PCP) that is defined in 3-bit field and drop eligible indicator (DEI). Those tags are defined in the 802.1p standard and in recent 802.1Q versions [24]. Even though PCP defines the priority for data and drop eligible tells if the packet can be dropped if there is congestion, it does not meet the requirements for time-criticality [8]. Figure 2.1 shows how 802.1Q header is inserted in an Ethernet frame.

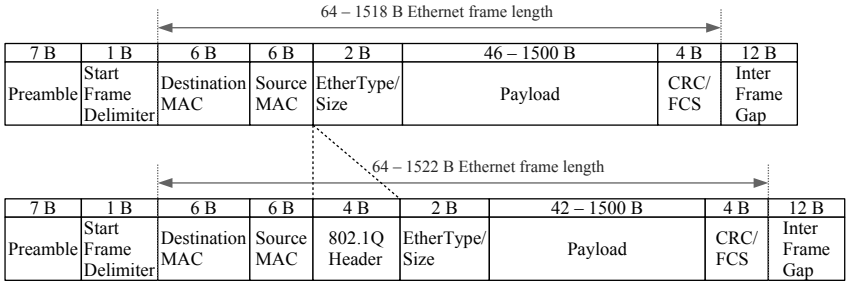


Figure 2.1: Insertion of 802.1Q header to a Ethernet frame [25]

Whenever there is a switch or other network node that does not work with 802.1Q, the header will get deleted. Therefore it is backwards compatible, but will lose some of the extra features the header is used to provide [6]. It must be also noted that the maximum size for Ethernet frame is grown by four bytes.

There are in total 4096 VLANs are available to use in a network. The VLAN tag is called VID, a VLAN identifier. A simple example of VLAN usage is shown in figure 2.2.

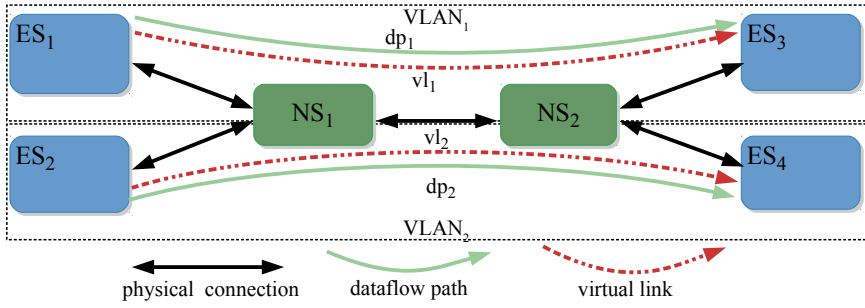


Figure 2.2: Two VLANs using shared switches

The two VLANs connecting ES_1 to ES_3 and ES_2 to ES_4 are called $VLAN_1$ and $VLAN_2$ respectively. Even though all data traffic is transmitted between NS_1 and NS_2, data from in different VLANs can be distinguished from 802.1Q header. This allows to have multiple networks using the same cables.

Because VLANs are able to act as separate networks, and priority can be defined for each frame, the $VLAN_1$ can demonstrate normal desktop computers and $VLAN_2$ some control units that need higher priority, for example industrial robots and a computer controlling it. This, however, does not mean that the protocol is deterministic, as there are no guarantees it meets real-time system requirements.

2.2 Technical details

Time-Sensitive Networking (TSN) addresses the lack of support for real-time systems in LANs [12]. It is an extension to IEEE 802.1Q and is known as 802.1Qbv. TSN is designed to work when traffic contains up to 75% time-critical data, the rest being best-effort [28][8]. Hence, it allows one network to replace several networks with different requirements and, therefore, provide simpler network solutions and cut costs.

Time-Sensitive Network task group was formed from Audio / Video Bridging Task Group. The original need to provide low latency for streaming services was broadened to have deterministic protocol for industrial and automotive use [12].

2.2.1 Prioritization

In order to be able to prioritize data transmission, there has to be certain rules in place. For 802.1Q the prioritization rules are made by traffic types and what the data is used for.

Table 2.2 shows the different priorities and what kind of traffic the priority is assigned to. There are a maximum of eight priorities as the tag is defined by three bits ($2^3 = 8$).

The recommended table of priorities by 802.1Qbv in table 2.3 shows that if there are less than eight available traffic classes, priority will be adjusted. All the adjustments are done for the node in question and result to lower priority and not higher due to the fact that priority zero is the default.

Table 2.2: Priority levels [11]

Priority	Traffic types
0 (lowest)	Background
1	Best effort
2	Excellent effort
3	Critical applications
4	Video less than 100 ms latency and jitter
5	Voice less than 10 ms latency and jitter
6	Internetwork Control
7 (highest)	Network Control

Table 2.3: Suggested priority table

		Available traffic classes							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Priority	0 (Default)	0	0	0	0	0	1	1	1
	1	0	0	0	0	0	0	0	0
	2	0	0	0	1	1	2	2	2
	3	0	0	0	1	1	2	3	3
	4	0	1	1	2	2	3	4	4
	5	0	1	1	2	2	3	4	5
	6	0	1	2	3	3	4	5	6
	7	0	1	2	3	4	5	6	7

Data transmission prioritization is done in an outbound port. A detailed model with all different parts of an outbound port is shown in figure 2.3 as are the steps for how frames go through the port. These steps are described more thoroughly below.

Inputted data comes in and is placed to a outbound port. The data goes through a queue selection unit (a) and is placed in the right queue (b). Transmission Selection Algorithm (TSA) selects (c), which data to take from the queue and if the following gate (e) is open, it can be transmitted. Open gates are determined by gate control list (d). Transmission selection (f) selects which data is to be transmitted from data output (g).

As the figure 2.3 shows, there are multiple queues to accommodate different traffic classes. That makes maximum 8 queues in total, one for each traffic class. If there are less than eight queues, each queue can contain data from multiple types of traffic classes. TSA decides in which order data will be transmitted from the queue. Even though TSA has selected some data to be transmitted next, the gate following it can block the transmission. That makes it possible to

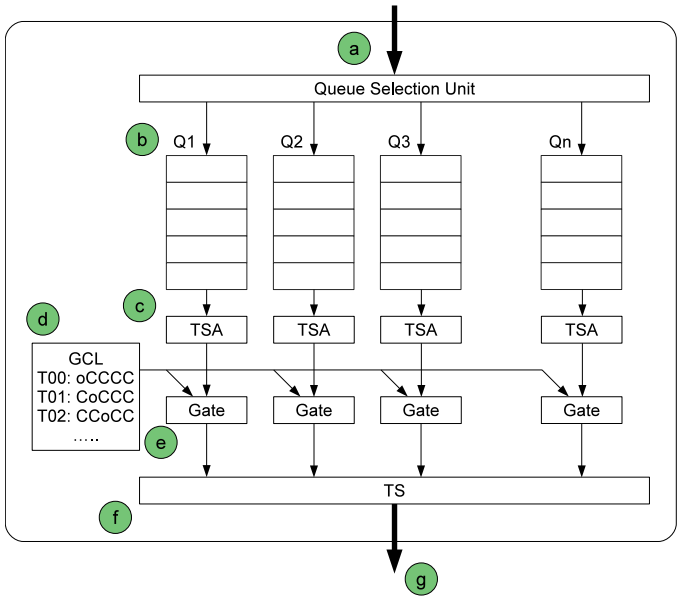


Figure 2.3: Outbound port (redo fig).

give priority to some other data than the highest priority traffic type.

The Transmission Selection Algorithm can be something else than a simple First In, First Out (FIFO). It can be a credit-based shaper (CBS) or a vendor specific algorithm [12].

Gates are the cornerstone of TSN [12][8]. Previous amendments to 802.1Q have included many prioritization improvements being the basis for 802.1Qbv [26]. It is possible to have each gate in whichever state, open or close. Gates are controlled by the gate control list. The operations in the gate control list are to open or close specific gates.

The standard does not, however, define how the gate control list should be used and therefore it is up to the industry to define how the gates are operated. If the implementation does not support enhancements for traffic scheduling, the gates do not exist an therefore cannot block the transmission.

The transmission select takes the highest priority frame available to be selected for transmission [12]. If a gate before the transmission select is closed, it will act similarly than that there is no data to be transmitted.

2.2.2 Time-aware shaper

Real-time systems have a need for highly predictable data delivery in terms of time. The latency and jitter for the transmission through all the nodes in the system has to be predictable as well. In an industrial environment control applications the control data can be transmitted on a repeating time schedule. For example if there is only one device sending time-critical data once a second, the pattern can repeat itself every second. Prioritization alone does not guarantee the control data can be transmitted at the right time, because if there is already lower priority data being transmitted, it will have to finish before anything else can be transmitted.

Because time-critical data transmission needs synchronization between hardware nodes, there has to be some kind of a protocol to take care about synchronizing clocks. TSN uses precision time protocol (PTP), with accuracy of one nanosecond [12]. It is used to synchronize the gate control lists with connected bridges.

The usage of time-critical data with synchronized gate control lists makes it possible to transmit data and have the correct gates open along the transmission route when needed. That will essentially create a clear communication channel for ultra low latency data transmission. The method is called cut-through forwarding or a protected window [8][12]. In time-triggered architectures the possible data transmission times are known beforehand, which makes it deterministic. Gate control list is used in order to achieve similar behavior for TSN. The gate control list is a repetitive pattern [12].

Protected windows need the data transmission for previous frames to be stopped before the window starts and for that reason a guard band (T0) can be placed before start of a protected window (T1), which is shown in figure 2.4. The end of the protected window (T2) reverses gate states back to what they were before the protected window [12].

The guard band time (T1 - T0) must cover the time the maximum size frame transmission takes if the outbound port is unable to determine how long it takes for the next frames to transmit. This will work even for the worst case, but is not an ideal solution as some frames may have to wait for until the end of the protected window (T2), even though they would be possible to be transmitted before the start of the protected window. If the outbound port can determine the next frame to be transmitted is less than the maximum frame size and it finishes before T1, the frame can be transmitted, thus increasing throughput and interfering less with not time-critical data [12].

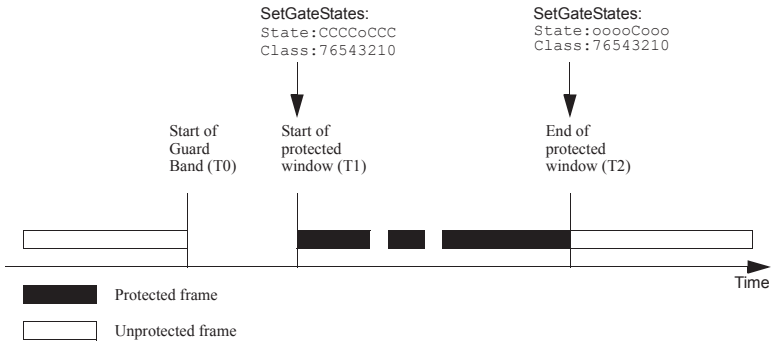


Figure 2.4: Protected window using gates [12]

The amount of time-critical data and protected windows have a limit since if all data is treated special, no data is special. Even though TSN is designed to be able to cope with 75% time-critical data, the specific requirements can affect the amount of time-critical data being able to handled.

2.2.3 Credit-based shaper

Although low latency can be achieved with protected windows and prioritization, in some cases removing jitter is more important. Protected windows cannot be used for all data either, because treating all the data as special treats no data special. Latency can be acceptable up to certain time, but in some cases jitter can degrade the service's quality as data frames can be considered as lost by an application [7]. When watching a video stream, some latency will not hurt, but jitter can degrade quality. If jitter is too high, some data can arrive too late and result in frame skipping leading to degraded quality of service.

In order to provide continuous stream of data for AVB frames (audio and video), the data is not necessarily to be transmitted as soon as possible. Because AVB frames have higher priority than many other traffic types (table 2.2), AVB data could end up transmitted in bursts resulting in blocking less important data unnecessarily.

To provide continuous stream a credit-based shaping can be used to transmit data more evenly. Credit-based shaper allows data to be transmitted if there are enough credits left. Credits are earned while it there is no transmission and consumed when data is being transmitted [5]. The basic idea of this is shown in figure 2.5.

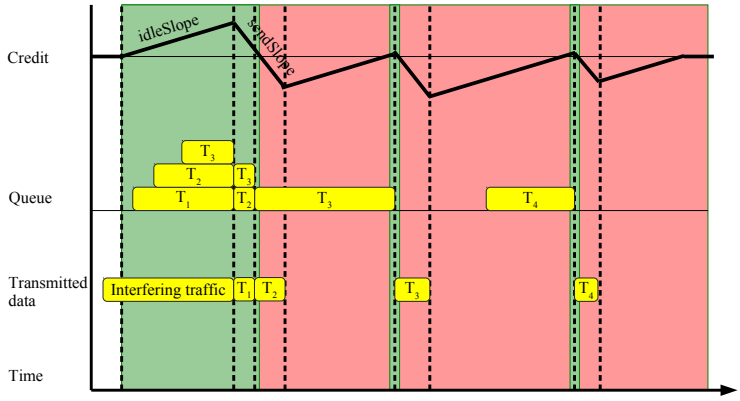


Figure 2.5: Credit-based shaper [5][28]

Whenever there are enough credits for transmission, the background is green and otherwise it is red. Frames are being sent using FIFO and as it can be seen, the frames are spread more evenly when transmitted than the arrival times.

2.2.4 Preemption

The standard IEEE 802.1Qbu defines preemption for frames. This makes it possible to pause transmitting a frame before its finishing and transmit something more important and then continue the paused frame [20][21]. A simple example of preemption is shown in figure 2.6.

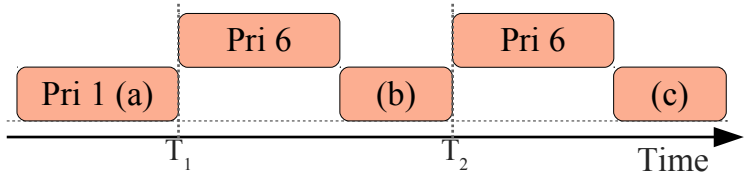


Figure 2.6: Preemption

When a frame is preempted, it can be continued where it was left after the higher priority frame has been finished [21]. The frame *Pri 1* is split in three parts as it is paused two times by higher priority frame.

Preemption makes it possible to utilize the network more efficiently. The small gaps between frames can be filled with best-effort traffic. It also gives higher priority frames the possibility to start transmitting before best-effort traffic frames are finished, which will reduce latency for high priority frames. This can be used in conjunction with credit-based shaper and protected windows.

2.3 Architecture

The network consists of two different kind of nodes, network switches (NS) and end systems (ES). End systems either send or receive data, but do not relay it. Network switches only relay data, but do not consume, create or alter it. Connection to other nodes is through outbound port. Each outbound port has its own queues and each port is connected to only one other node. Therefore there can be more than one outbound port in each node. The number of ports receiving and transmitting can be different from each other. One important part of TSN is a clock inside a node. It is used to synchronize gate timings between nodes to enable protected window feature [12]. A basic network is shown in figure 2.7.

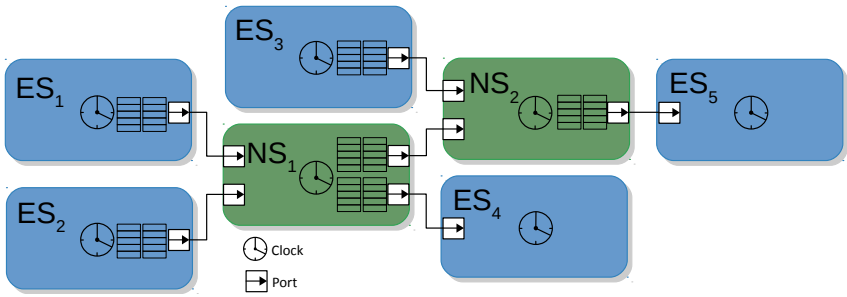


Figure 2.7: TSN network architecture example

All the sent data is considered to be received by other nodes. That leaves prioritization to be done in the outbound port. For an architectural point of view, this simplifies the way processes seem to happen as everything can be considered to be happening in the outbound port.

2.4 Example

The examples in this section are used to demonstrate how TSN and prioritization works. The first example focuses on how the frame travels through the system – it is not meant to show how scheduling is done. The second and third examples are focusing to demonstrate how frame prioritization works and what kind of impact protected windows have to the network and time-critical data.

2.4.1 Data flow

A simple example makes it easy to see how data is traveling through nodes. This example is simplified from architecture figure (fig. 2.7) to be more compact and easier to understand. Because there are different algorithms for TSA it is considered to be FIFO, although in this example it does not matter. Using gates it is possible to override priorities within outbound port and since how to use the gates is up to manufacturers, all the gates are open constantly. The example is shown in figure 2.8.

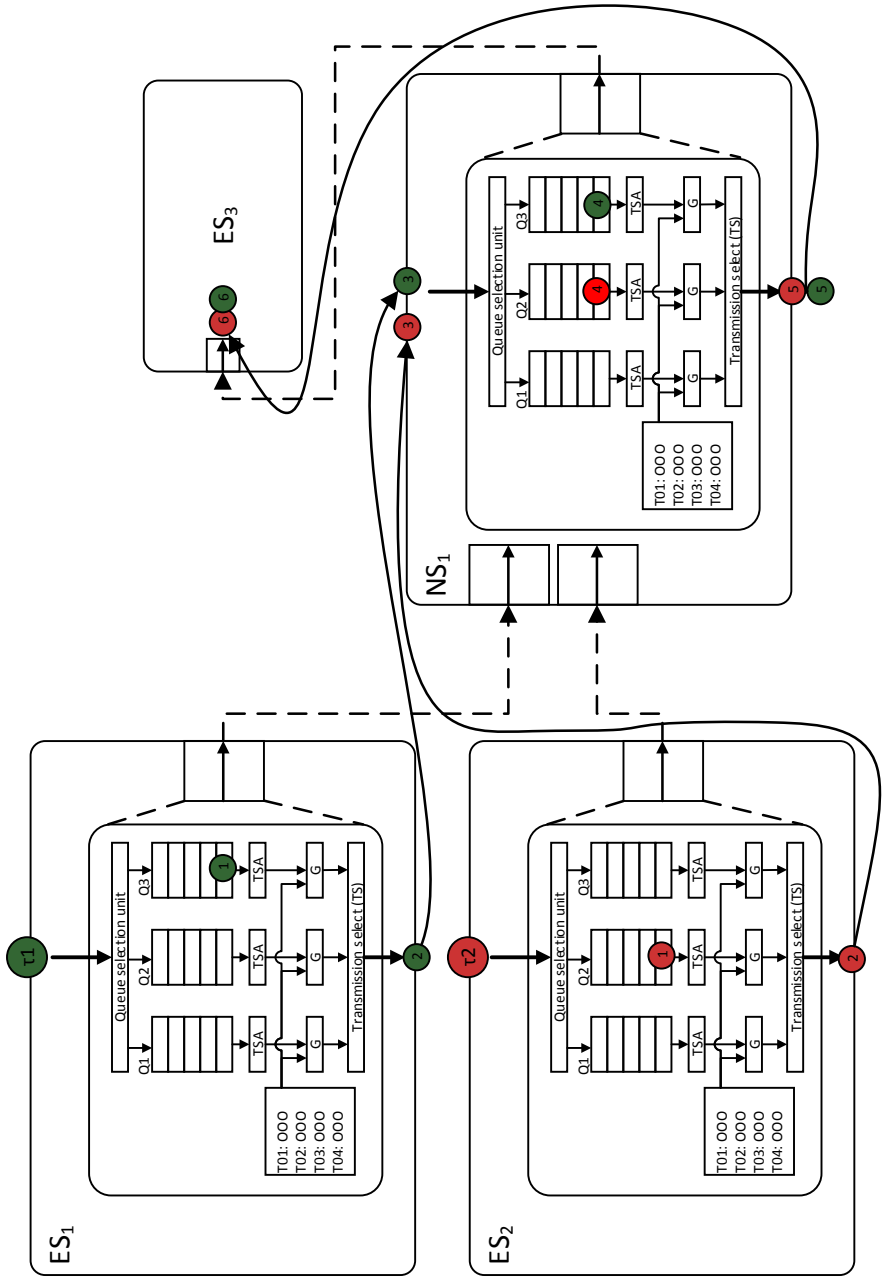


Figure 2.8: Data going through nodes

In the example, there are two data frames τ_1 (green) and τ_2 (red) with numbers to indicate the order the frames flow. Let us consider both are maximum size for one frame. Both of the messages are transmitted immediately from End Systems 1 and 2 to End System 3. On the route from ES_1 and ES_2 to ES_3, the data goes through Network Switch 1. ES_1, ES_2 and NS_1 show an enlarged outbound port with more details, as it is the main point in the standard.

When both messages arrive in NS_1, both messages are put into corresponding queues, considered arriving at the same time. When the messages are transmitted from NS_1, τ_1 will be transmitted before τ_2, because it is a higher priority message. This example is merely to show how the data frames are going through the nodes and not how everything is scheduled.

2.4.2 Scheduling

A more in depth example (with network built like in figure 2.2) is shown in figure 2.9. Let us define one unit of time as a *step*.

Table 2.4: Frame characteristics

ID	Priority	Interval	First arrival	Transmission time
1	1	10	0	3
2	3	10	2	2
3	5	6	6	2

The characteristics for example frames have been defined in table 2.4. The values are chosen to show why problems with time-criticality may arise even with priorities defined.

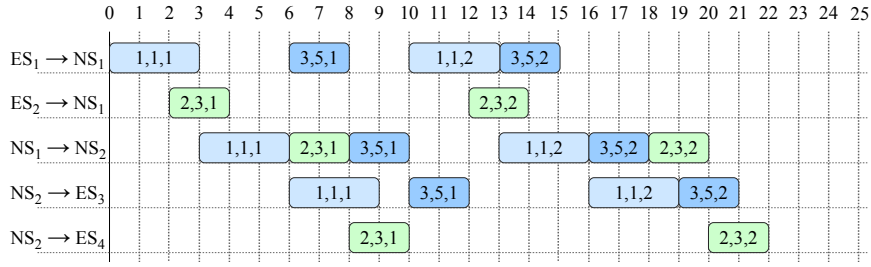


Figure 2.9: Schedule without gate operations in a simple network

Even though the frames have different priorities, their latency is not more stable with high priority. The lowest priority frames *ID 1* have stable latency at nine

Table 2.5: Results from figure 2.9]

ID	Arrival	Finish	Latency
1,1	0	9	9
1,2	10	19	9
2,1	2	10	8
2,2	12	22	10
3,1	6	12	6
3,2	12	21	9

steps, *ID 2* frames have latency from eight to ten steps and the highest priority *ID 3* has latency from six to nine steps.

Now let us consider the gates are featured in the outbound port and frames with *ID 2* are considered time-critical and protected windows are used to guarantee low latency. All the other characteristics of the frames are kept the same. The result is shown in figure 2.10. The protected window in this example is used only for traffic class 3 (priority 3), which affects only *ID 2* frames.

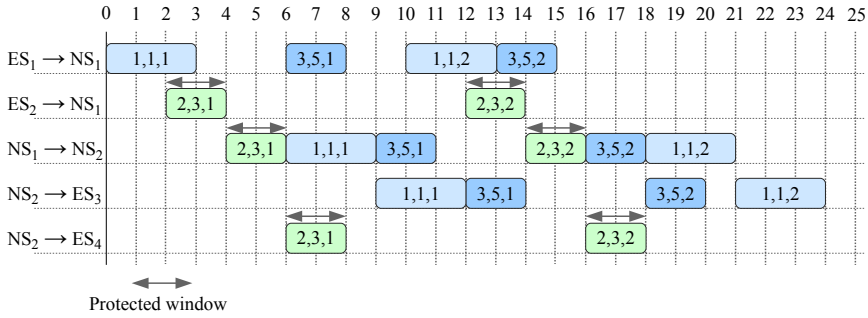


Figure 2.10: Schedule with gate operations in a simple network

Table 2.6: Results from figure 2.10]

ID	Arrival	Finish	Latency	Difference
1,1	0	12	12	+3 (33%)
1,2	10	24	14	+5 (56%)
2,1	2	8	6	−2 (25%)
2,2	12	18	6	−4 (40%)
3,1	6	14	8	+2 (33%)
3,2	12	20	8	−1 (11%)

The differences are significant even in this example in a small network. One notable observation is that the total time to transmit all frames is increased with

gate operations – from 51 to 54 steps – with great expense of low priority traffic. However, the objective to treat *ID 2* frames as time-critical with protected windows works. The latency is stable 6 steps, which is the optimal time for transmission from ES_2 to ES_4.

2.5 Summary

Time-Sensitive Networking is an enhancement for existing standards. Therefore it is backwards compatible, just like previous prioritization and bridging standards and amendments. The architecture and prioritization handling are simple and easy to represent. The examples show that protected windows will make TSN suitable for time-critical data in theory. Due to the fact that TSN is an enhancement for existing standards it eliminates the need for different networks even for real-time systems. This results in cost efficiency and simplified infrastructure.

Other features – in addition to protected windows – can make the network perform even better. Credit-based shaper and preemption are good examples of this. The use of credit-based shaper will let lower priority data to be transmitted more evenly as well as removing bursts of higher priority data. Preemption will make it possible to transmit lower priority data at any given time and pausing it for higher priority data. This will result in more efficiently utilized network.

Chapter 3

Modeling and simulation

This chapter contains discussion about modeling and simulation a system. Simulating a system requires understanding different simulation paradigms, discrete-event and continuous-event, and what characteristics should be taken into account. The simulation has to provide also data from the simulation for later analyzes and comparison. One of the main concerns is that the simulated system should be realistic but at the same time it should be possible to simulate – the system design plays a major role in how realistic the simulation will be.

3.1 System

A system is the collection of components to be observed. The components are connected or dependent of other components – thus creating a system. An example LAN system is shown in figure 3.1. It contains computers connected to other computers and a server via switches. The system is observed by the state parameters. These parameters can be for example the number of messages in outbound queue and delivered message information, such as lag or missed deadlines. The actual parameters are of course chosen also depending the system and the problem being studied.

Discrete-event and continuous-event systems are the two main types. This LAN

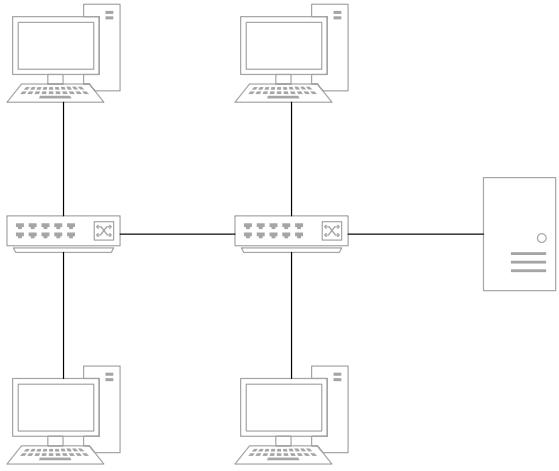


Figure 3.1: An example of a simple system.

system changes state whenever message is created to be transmitted or delivered. The message can be either waiting to be transmitted, being transmitted or finished and the state change is instant, making the system discrete-event. Continuous-event system would have state changes constantly over time, which works rather for example physical movement of an object.

3.2 Model

Systems can be studied in different ways depending on the needs and the system size for example. It is either possible to build a system and run experiments in the system or create a model of it [17]. The bigger the system is, the more resources are needed to actually build it and therefore a model is more feasible in many situations [4]. Model is also the only feasible way to experiment a system if the technology does not exist yet or when forecasts are needed. All the different ways of studying a system are shown in figure 3.2.

The model is represents a simplification of the system with certain assumptions. It is created for studying how the system works if everything works as assumed. That makes it a scaled down ideal system without any external distractions or failures and it only includes the parts of the system having effect to the problem being studied.

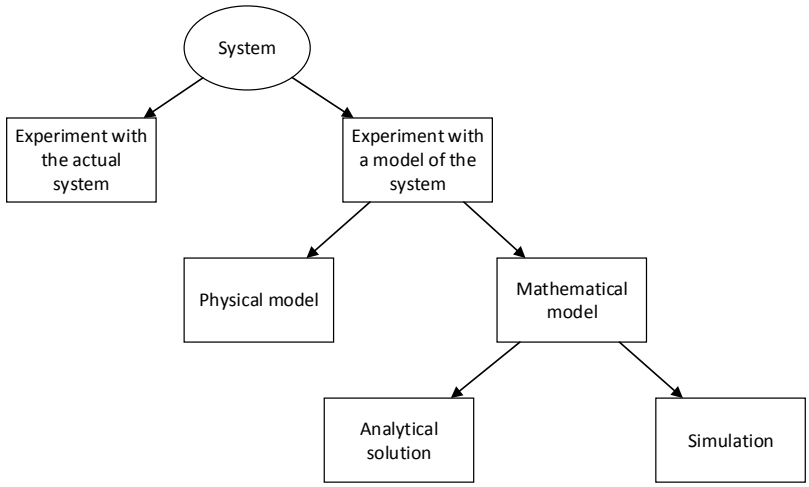


Figure 3.2: Different ways to study a system [17].

The model type depends on the studied system and how simplified it is assumed to be. The different types of model are: [4]

- dynamic or static
- deterministic or stochastic
- discrete-event or continuous-event

Simulating a dynamic model has "evolution over time" whereas static model is model not dependent on time [4].

The difference between deterministic and stochastic model is random elements. Deterministic model does not have any random elements and the results from each simulation run should be exactly the same, which makes it predictable. Stochastic model includes some random elements. In the LAN system (figure 3.1) the randomness could be messages appearing at random times or with random properties.

Discrete-event and continuous-event models are discussed more thoroughly in the section 3.3. The system in the previously shown figure 3.1 is a discrete-event system.

With the definitions for different models, TSN model should be built to be

dynamic, stochastic and discrete-event. A deterministic model can be used to make sure the simulator works correctly, but certain random events should be used for proper results from the simulator.

3.2.1 Verification and validation

The model is not the same as a real system so it must be verified and validated. Verification and validation try to answer to the questions "are we building the product right?" and "are we building the right product?" respectively and the product being the model [4].

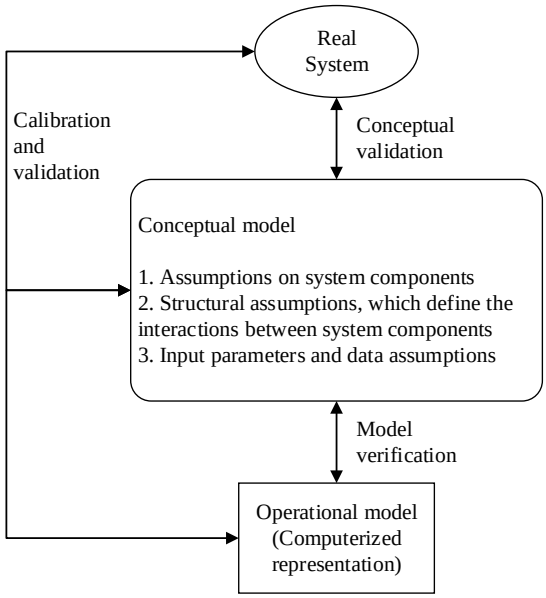


Figure 3.3: Model construction [2]

Validation means comparing the real system to the conceptual model. It answers if the model is built right to model the real system. Verifying the model is checking all the features that are designed in conceptual model are actually implemented and working [4].

Validation process is used to check if the model represents the real world system properly. The model cannot represent everything as it is only a mathematical model (see figure 3.2) and therefore an approximation, not a complete system [4].

The model should therefore contain only the characteristics needed to solve the problem in question. When the model is refined, it is called calibration and it will be done until the model is validated.

The verification process can be done by checking for example the output or creating flow charts [4]. To check the output data, a simple example can be done by hand and compare the results.

3.3 Simulation paradigms

There are two main paradigms to simulate a system, discrete-event or continuous-event [4]. In reality the system could have some features from both of the systems. In discrete-event systems the state changes in an instant at certain point of time whereas in continuous-event systems state changes constantly over time.

3.3.1 Discrete-event

Queuing for example in a bank works as discrete-event model [2]. People appear instantly at certain points of time into queue. When they are called for being served, they are removed from the queue. The state changes therefore happen only when people come into the queue and when they leave it. The state changes are also immediate – the queue size is limited by natural numbers (0, 1, 2, ...) without in-between states. The figure 3.4 shows discrete-event simulation time advancing.

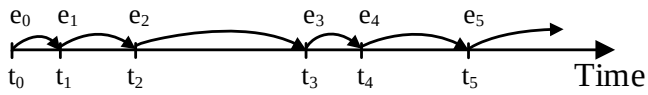


Figure 3.4: Discrete-event time advancing [4]

As the figure shows, the time advances in steps of different size. Because there is nothing happening between events, the time can be advanced directly to the next event's time, which makes simulation a lot faster. The events are marked as e_n and time points are marked t_n.

3.3.2 Continuous-event

A good example of a continuous-event model is throwing a ball. Its position and speed changes constantly over time. Therefore the state changes all the time.

Depending on the system and resources, it can be unfeasible to get the data at every point of time and from the observers perspective the states change at discrete jumps [4]. The figure 3.5 shows continuous-event simulation time advancing.

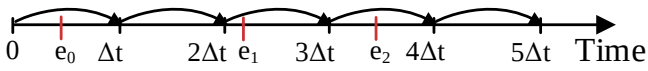


Figure 3.5: Continuous-event time advancing [4]

In the continuous-event simulation, the time advances in predefined steps of time. Events can happen during between the steps of time, even if the step would be extremely small. Again, the events are marked as e_n but time is shown as Δt, where Δ is the size of time increment.

3.4 Simulation

Simulation is using the model to imitate a real system [4]. There are different ways to do simulation, from simulation software to usage of general programming languages [18]. If general programming languages are being used to build the simulator, there is a need for auxiliary functionality for producing to produce for example stochastic features and data presentation. The simulator should provide enough data related to the problem in question to understand how the real system would behave.

3.4.1 Input data

An important part of modeling is to define input, such as frame arrival time, size, priority and deadline. As the model uses stochastic simulation, there is randomness in inputs. However, there is also control data to be transmitted, which is by nature deterministic rather than random.

There are three different situations regarding the availability of data to use in the simulation model [13]:

(i) no real-life data available

(ii) data of real-life output available

(iii) data of input and output available

Each of the three cases lead to different amount of evaluation and validation. If there is no real data available, it is hard to validate the model properly [13]. The data has to be generated with with theoretical randomness, for example normal distribution. This can lead to unrealistic results and the simulation being merely a best guess.

When there is real-life output data available, it is possible to generate limits for the data. That way the input values can be closer to the real-life values, but possibly ignoring rare cases outside limits [13]. Compared to case (i), the distribution of values can be fit more realistically.

Simulation is called trace-driven, when there is real-life input and output data available. This compared to cases (i) and (ii) gives advantage for validating simulator output by comparing it to real-life output. Using trace-driven simulation it is also possible to run the simulator multiple times with same input.

The simulator in this project uses trace-driven simulation. Since some data is possible to use to compare the results, it is also easier to validate the simulator working correctly and later make optimizations.

3.4.2 Data analysis

Analyzing the output produced by a simulator is an important part of simulation. The data accumulated during simulation has to be analyzed with paying attention to the termination conditions. The simulator can be either terminating or non-terminating.

Simulation is terminating if there exists values in state variables or value of time, which specifies the length of the simulation [4]. A value defining termination could be, for example, a specific data frame reaches the destination. Terminating simulations tend to make transient behavior more dominating [4].

Non-terminating simulation is run until the state variables are considered stable [4]. Transient behavior is therefore absent as the simulator will be run long enough for not to be affected by extreme cases.

The simulator for TSN needs to find a stable state as the extreme cases can vary a lot depending on the network and the amount of traffic. Different extremes are created by using stochastic model to include randomness.

3.4.3 Creating randomness

In order to create stochastic simulator, there has to be some randomness. To achieve this, the simulator has to use randomly generated data and this can be achieved with random number generator (RNG). The result should be a number that is generated uniformly in range $[0, 1]$ without correlation between generated values [2][4][22]. The default Math.Random() -function in Java was deemed to fulfill the requirements [22].

Random numbers should not be dependent on already generated ones and they should be distributed equally. Java random number generator fulfills this requirements as well [22].

Chapter 4

Simulator design and implementation

This chapter explains the implementation of TSN-simulator. It includes the requirements to achieve objectives mentioned in section **??** and to simulate model defined in chapter 3. The design and implementation are in separate sections to explain the design choices and how everything was implemented.

4.1 Requirements

The objective is to simulate TSN and evaluate design choices for different algorithms for scheduling. The requirements for the simulated are:

- Generate messages from a file to allow optimization to be done with given message characteristics
- Generate a network from a file
- Generate virtual links from a file to determine routes for messages
- Model action and event-oriented simulator

- Model both terminating and non-terminating simulator
- Output average and worst-case latencies for non-time-critical messages
- Output gate change event times
- Output times for messages to arrive and leave nodes
- Validate simulator

4.2 Design

The simulator has to be able to handle different kind of setups in the system, provided by files. Each network switch can be connected to any number of other network switches and end systems and each end system is connected to one network switch. The routing and messages are determined in files as well.

4.2.1 Assumptions

Due to the scope of this thesis, the simulator is designed to simulate only fully functional network. That means all the data is always correctly delivered and all the defined links between nodes are available. The data is routed using virtual links. Considering assumptions and requirements, the simulator has the following characteristics:

- No packet losses or other failures
- One message per frame
- The clock is synchronized globally in the network
- The network is homogeneous
- Time-critical messages appear exactly as scheduled
- Time-critical messages flow through the system using protected windows without any delays
- Time is incremented at minimum 80 ns, which is the time to transmit 1 byte (100 Mbps network)
- Routing is static

- There are no delays in the hardware

This results in very high-level abstract simulator of Time-Sensitive Networking. To simulate TSN more realistically, there are multiple assumptions that have to removed and implement features.

4.2.2 Program parameters

The simulator needs data in order to work so it must be given before running it. Because the simulator does not have a graphical user interface and the data should be changeable but still constant if needed, the data is read from files. The filenames are given as command line parameters. The list of the parameters is:

finalize this whe

- -m <messagefile> – a file to define messages and their characteristics
- -n <networkfile> – a GraphML file to specify the topology of network
- -v <virtual link file> – a file to describe virtual links in the network
- -o <output directory> – path to a directory to place all the output files
- -t time – maximum time to run simulation (in microseconds) or micro?
- -d difference – maximum difference to denote stable state (1.01 is 1 %)
- -s strict mode – do not allow overlapping protected windows (1/0) default 1

Using these command line parameters it is easy to create different scenarios and keep the data untouched between runs.

4.2.3 Simulator output

The output files will be placed in previously given output directory:

finalize this whe

- data.txt – the file containing the statistics for messages

- data.csv – the file containing statistics for messages in CSV-format
- data_all.csv – the file containing statistics for all message instances in CSV-format
- history.txt – the file containing all transmit times for each nodes for each message
- graph.svg – the file for checking how messages are being sent in SVG-format

The data includes enough information to allow further analyzing and comparison. The files will have a suffix denoting the modes used. Therefore data.csv can become for example data_qbv.csv or data_qav_qbv.txt.

4.2.4 Data routing

There are different ways to create routing between the starting point and the destination (sender and receiver). Because routes are defined in an input file, the routes for this simulator are defined as static. Virtual links are used to define the routes between end systems in the network. The network connections are predefined from a network graph file. An example virtual link is shown in figure 4.1.

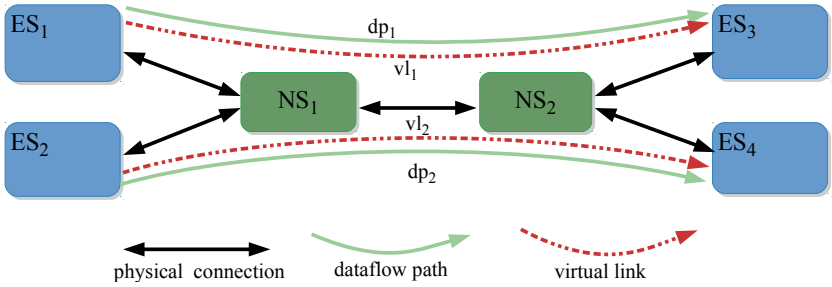


Figure 4.1: Virtual link in a network

There can be any number of virtual links as they merely define a subset of a network. This subset is used when frames are transmitted and need to know the next node.

4.3 Implementation

The simulator was continued from existing project in Java started in Eclipse. Work was needed to feature gate-events and prioritization, for example. The gates are used to create protected windows.

Before any simulation is done, the network and messages have to be generated. After all the messages have been generated, the simulator creates gate events. Because the simulator has a maximum running time defined from the beginning and all the messages are generated before running the simulation, the gates are defined before simulation as well.

Using Java allowed to have all the different entities in their own objects as Java is object-oriented programming language. This way it is easy to separate all the parts in an outbound port from each other (figure 2.3).

At first, the gates are created to be open all the time and after that, the protected windows are created for time-critical frames. A simplified UML-diagram of the simulator is shown in figure 4.2.

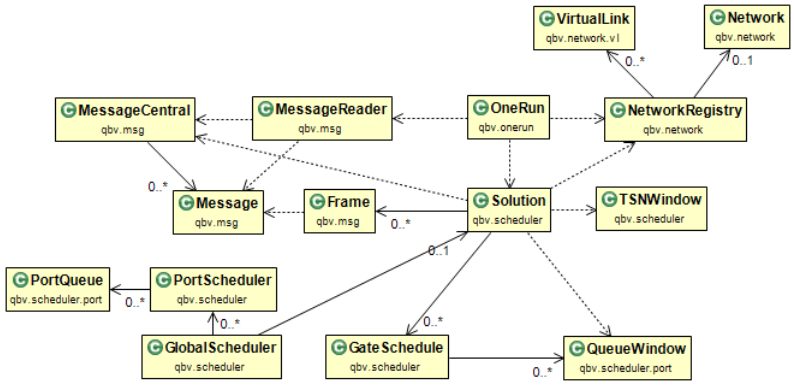


Figure 4.2: Simplified UML-diagram of the simulator

The class *OneRun* is the main program of the simulator. It reads the command line parameters and passes them to appropriate classes to be handled as well as starts the simulation in *GlobalScheduler*.

Before any simulation is done, the network, virtual links, messages and gate events need to be generated. The network and virtual links are hold in *NetworkRegistry*, which takes care of generating the network and is explained in more detail later.

Singleton pattern was used to refer to the correct objects and to have only one instance of some classes. The classes to have only one instance are *MessageCentral*, *GlobalScheduler*, *GlobalTime*, *PortHistoryKeeper* and *NetworkRegistry*. The main reason is to have messages, time, network and the scheduler in the one instance of the class.

4.3.1 Messages

Message information is read by *MessageReader* from a predefined file, to create instances of *Message*s to be stored in *MessageCentral*. The number of messages to be created is:

$$n = \frac{maxTime}{msgPeriod} \tag{4.1}$$

The number of messages can become very high if the maximum runtime is defined too long – it should be noted that the simulator needs to be changed to generate messages dynamically if longer runs are needed. After all the messages have been created, they are assigned in frames.

The frames are kept in *Solution*. Because all the messages are essentially similar – time-critical messages are defined by a certain traffic class and given clear route with protected windows – there is no need to create different frame or message types.

The messages that are not time-critical are generated to have random release times calculated from their period and transmission time. At first, the random time is calculated.

$$jitter = random * (msgPeriod - msgTransmissionTime) \tag{4.2}$$

Where *random* is a uniform random number between 0 and 1. The appearing time for the message is calculated as follows:

$$appearTime = (n * msgPeriod) + jitter \tag{4.3}$$

Where n is the number of message instance from equation 4.1. This way the messages will appear randomly within certain limits but a new message will not

appear before the previous one could have been transmitted in theory.

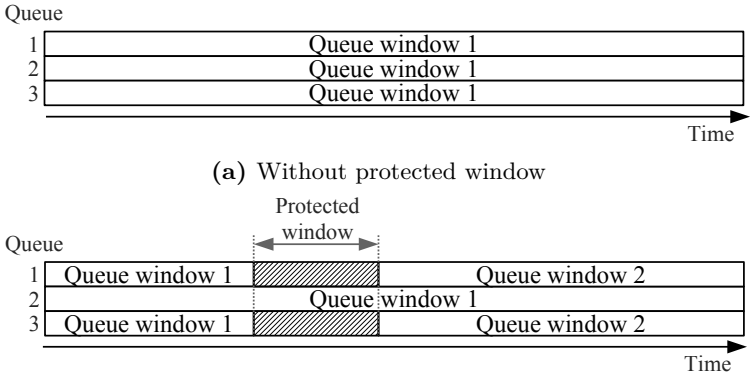
4.3.2 Protected windows

The *GateSchedule* is created to have windows of time when the gate is open. Each port has an assigned *GateSchedule* and it contains a list of *QueueWindow*s. *QueueWindow* has information for which queue it belongs and when the window opens and closes.

By default there are *QueueWindow* for all gates from the beginning to the end. This creates open gate all the time. Afterwards all the gates are closed for the time defined in *TSNWindow*, except the one needed for time-critical frame. This creates a protected window. The creation of protected window is straightforward:

1. End all queue windows to the start of protected window, except for the protected window gate.
2. Create new queue windows to start at the end of protected window until the time the old queue window was open.

The process is shown in figure 4.3 with the starting point and end result.



(a) Without protected window

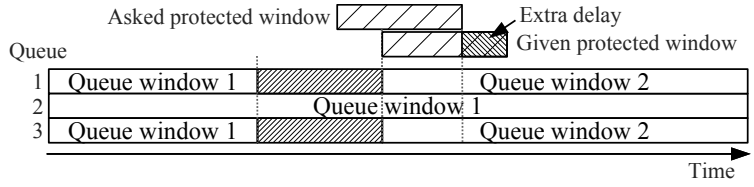
(b) With protected window

Figure 4.3: Creating protected window

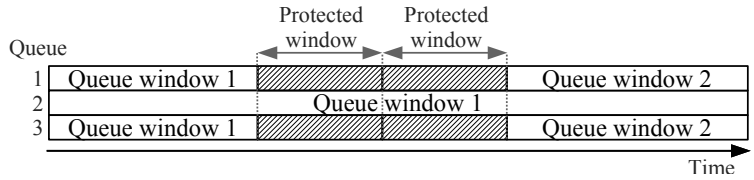
In this implementation there is no need for guard bands – the port knows how big messages can be transmitted before protected window and calculates if it

is possible to transmit the next message. It can even transmit lower priority message if any higher priority messages cannot be transmitted in time.

Overlapping protected windows will be adjusted by moving the later one at the end of the first one. This will, however, add significant delay and should be taken care when designing the timings for time-critical messages. Therefore there is a command line parameter to disallow overlapping windows and if one exists the simulator will not run, but will give a warning. Figure 4.4 describes the creation of overlapping protected window.



(a) Adding overlapping protected window



(b) Finished overlapping window

Figure 4.4: Creating overlapping protected window

Overlapping protected windows create extra delay for the later protected window. It is always the later window, which will be moved to a later point of time. The extra delay is the amount of overlap the windows have.

4.3.3 Network

The network is roughly divided in network and virtual links. A more detailed UML-diagram of the network is shown in figure 4.5.

The class *NetworkRegistry* holds information about *Network* and *VirtualLink*s. These are divided into classes to generate the network (*NetworkBuilder*, *VirtualNetworkBuilder* and *NetworkNodeFactory*), nodes (*NetworkNode* and *VirtualNetworkNode*) and links (*DataflowLink* and *VirtualDataflowLink*).

At first, the “real” network is created to *Network*-class to contain links between

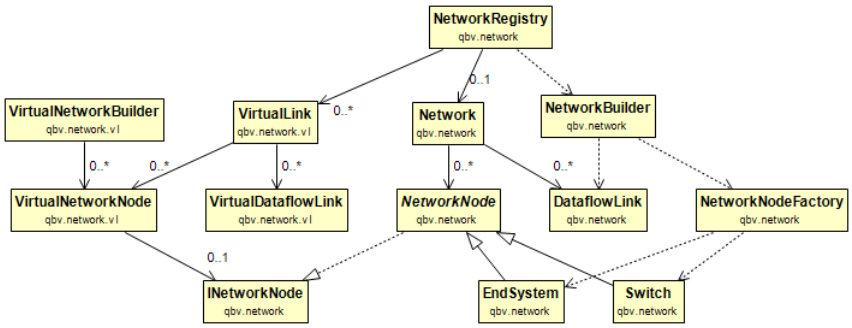


Figure 4.5: UML-diagram describing network

nodes, which can either be an *EndSystem* or a *Switch*. The actual creation is assigned to *NetworkBuilder*, which returns an instance of *Network* and to *NetworkNodeFactory*, which returns an instance of *NetworkNode*. *DataflowLinks* have the information of how many queues the outbound port has and what is its speed.

The information stored in *Network* allows traversing the network and the links between different nodes. This is the basis to create *VirtualLink*s on top of it.

4.3.4 Virtual links

The virtual links are stored in *VirtualLink*. They are used to create static routing on the network, which the messages will use. The links are created by reading a file defined in command line parameters (section 4.2.2) and creating a "virtual network" on top of the "real" network, which is basically a subset of the whole network. During the creation of virtual links, it is checked that the network nodes are connected to each other in the real network.

4.3.5 Simulation

After the network and messages have been generated, it is possible to start the simulation itself. The core of simulation is *GlobalScheduler*. The UML-diagram of the most important classes during simulation is shown in figure 4.6.

GlobalScheduler handles advancing time and using *PortScheduler* to transmit frames. *PortQueue* is essentially a linked list of *Frame*s, which gives the next

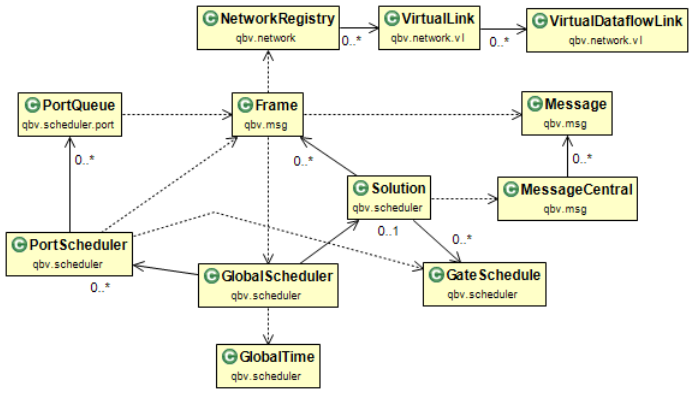


Figure 4.6: UML-diagram describing simulating

frame to be transmitted to the *PortScheduler*. Whenever frame transmit is finished, it is moved to the next node's (*PortScheduler*) queue (*PortQueue*). The next node is determined from *VirtualDataflowLink*. A global clock to keep the network synchronized is provided by *GlobalTime*.

4.3.6 Output data and statistics

Data for events in ports and frames are kept in *PortHistoryKeeper*. *PortHistory* keeps the history of frames flowing through ports and *FrameHistory* keeps track of frame latencies through from the beginning to the end. This data is used to provide both text files and a graphical SVG-file to see how the frames flow through the system. The components to take care of statistics are shown in figure 4.7.

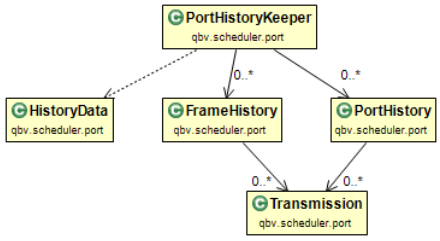


Figure 4.7: UML-diagram describing output data generation

The class *HistoryData* is used to contain all ports transmission data to construct the SVG-file. It has no methods except it implements *Comparable* to make it

possible to sort data by the port identifier for easier data manipulation and organizing.

One of the outputs is a CSV-file (data.csv), which contains information about each message. CSV-files are easy to use in different programs and make graphs or calculations from it. The data inside the file is constructed as in this simple sample file:

```
"Frame","Best-case delay","Average delay","Worst-case delay","Jitter"
"tc2","170880","170880","170880","0"
"tc1","170880","170880","170880","0"
"av2","143600","182326","315680","32519"
"av1","95600","171553","319200","39850"
"be1","88320","136215","266800","28714"
```

Even more detailed CSV-file (data_all.csv) of messages is saved to see the exact details of each instance of all the messages. It contains latency, start time and end time for each instance.

This data allows analyzing for bottlenecks and effects for time-critical message transmitting in the network. The analyzes can include jitter for example and deadline misses. However, as the network is assumed to work correctly, there is no packet losses as these do not occur in the simulation.

4.4 Running the simulator

As the simulator is discrete-event dynamic simulator type, it makes sense to advance time to the next event. The loop that runs the simulation works as follows:

1. Place messages released at this moment to their initial ports
2. $step = 80ns$[1]
3. For each first[2] frame in each *PortQueue*, determine if the frame can finish before the gate closes
4. For all the frames that can finish, determine the time for first one to finish ($step = frameFinish - now$)

[1]Time it takes to transmit one byte
[2]TSA selected frame

5. Check if any new frames appear before $now + step$
   - assign $step = nextAppear - now$
6. Execute frames
7. Advance time by amount of $step - 80ns$[3]

By skipping all the unnecessary steps of time, the program works much faster, depending on the input data. Therefore it can be used to run for a longer time and find an even "more stable" state.

However, this design choice for the simulator is not completely only event-based. The simulator merely skips a lot of clock cycles, but in principal it works exactly as it would work by incrementing time one nanosecond at a time. The main simulation loop is shown in figure 4.8 as an activity diagram.

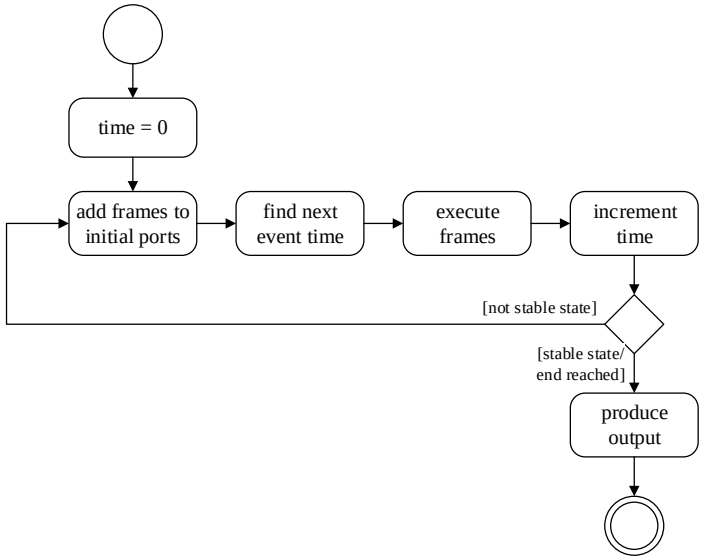


Figure 4.8: Simulator scheduler activity diagram

The main loop for simulator looks fairly simple and the more complex parts belong to the frame execution and finding the next event time. The following paragraphs describe each activity more closely.

[3]Frame is removed one cycle earlier, time advance is minimum of 80 ns

4.4.1 Frame initialization

Frames are added to their initial ports when frame's appear time matches the global time. The frame is placed to the correct queue determined by frame's priority and the number of queues in the port as suggested in the table 2.3. The activity diagram for placing frames to their initial ports is shown in figure 4.9.

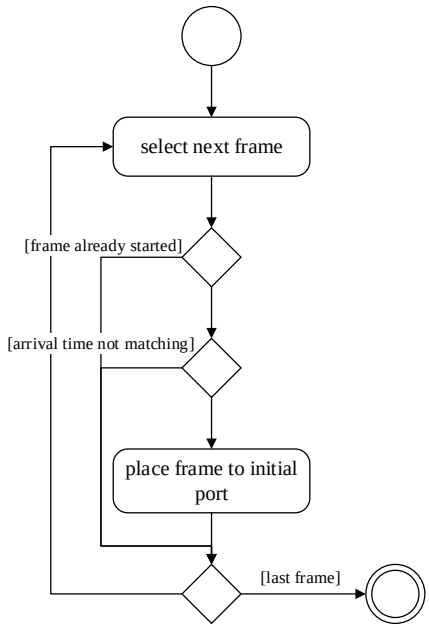


Figure 4.9: Simulator scheduler activity diagram

This loop is executed in the beginning of scheduler loop as shown in figure 4.8. All the frames are checked if they have not been started and their appear time matches the global time and placed to their initial ports. This design choice was taken in order not to fill the ports with waiting frames – as they are created at the beginning – but keeping them in a storage (*MessageCentral*, fig. 4.6) to release them at the right moment.

4.4.2 Finding next event

In order to skip the steps of time when nothing happens, it is mandatory to find out when next event happens. The next event can be something from the list:

- Frame arrives
- Frame starts sending
- Frame finishes

As frames are the objects being moved from one node to the next one, they have information required for simulating the transmit. The attributes for frames are shown in figure 4.10.

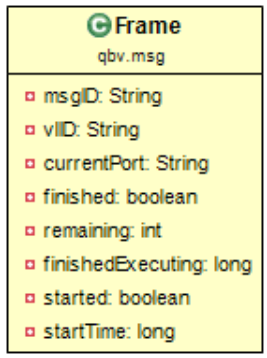


Figure 4.10: Frame attributes

The attributes contain critical information for frame transmission. The data in the attributes contains:

- msgID – ID for the frame instance
- vlID – Virtual link id to define the path
- currentPort – The name of current port (link between nodes)
- finished – Defines if the frame has reached the destination
- remaining – Remaining time for transmission to the next node
- finishedExecuting – Defines the time when the frame reached the destination

- started – Defines if the frame has been placed to the initial port
- startTime – Defines the time when the frame was placed to the initial port

The time for frame arrival or finishing and starting sending can be the same if there is no other frames in the port and there is no protected window coming. However, the events are completely separate. The activity diagram to find out the next event time is shown in figure 4.11.

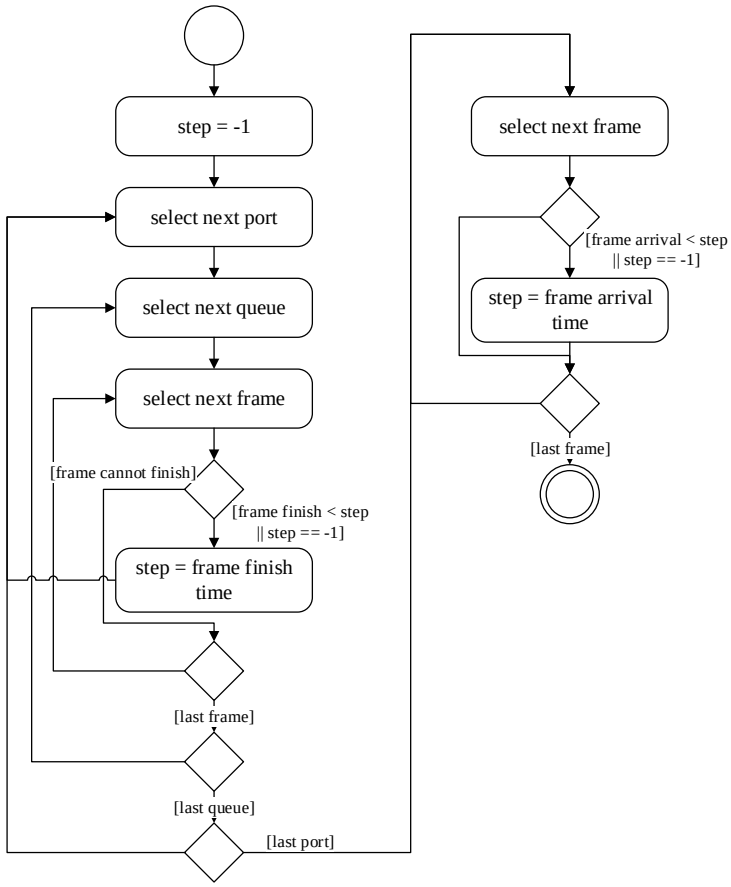


Figure 4.11: Finding the next event

The left side of the figure is to determine the next frame finishing event and the right side is to determine the next frame arrival time. Events for frames to start sending are always right after frame arrival or frame finishing – protected

windows are always optimal length and therefore do not block other frames for no reason – and therefore the events need not to be found.

One notable detail is that frame cannot finish if a protected window is coming before the frame could be transmitted. Then the next frame can be selected to try to send. If no protected windows exists, only the amount of ports frames (the first frame from each node) are checked for next event.

There are of course some optimizations, such as breaking from the loop if *step* is the minimum tick time at any point. These are not shown in the figure to keep it more clear and because these do not need change the logic.

It is also impossible for *step* to be anything else than multiply of minimum tick and that is handled before any time increments and frame executions are done.

For other activated features (credit based shaper, preemption support), the next event checking is modified similarly than the frame execution explained in the following section.

4.4.3 Executing frames

One of the core parts of the simulator is transmitting frames according to the data given to the simulator. Executing a frame has four different parts:

- Start sending the frame
- Reduce *step* amount of time from remaining transmit (fig. 4.10)
- Finish current transmit
- Place the frame to the next node if not finished

In order to execute frames, the simulator has to determine which ones to select for it. The logic is similar than in finding the next event – it is the same frames to look for – and therefore the activity diagram shown in figure 4.12 looks very similar. The same function is used to check if a frame can be executed.

Whenever a frame is executed, the next frame is selected from the next port. There cannot be any overlapping frame transmitting happening in one port. If the frame is executed, it is also checked if it has no remaining time for transmit left. When there is no remaining time left – frame finishes event – the frame will be placed to the next port or marked finished if there is no more nodes left to go.

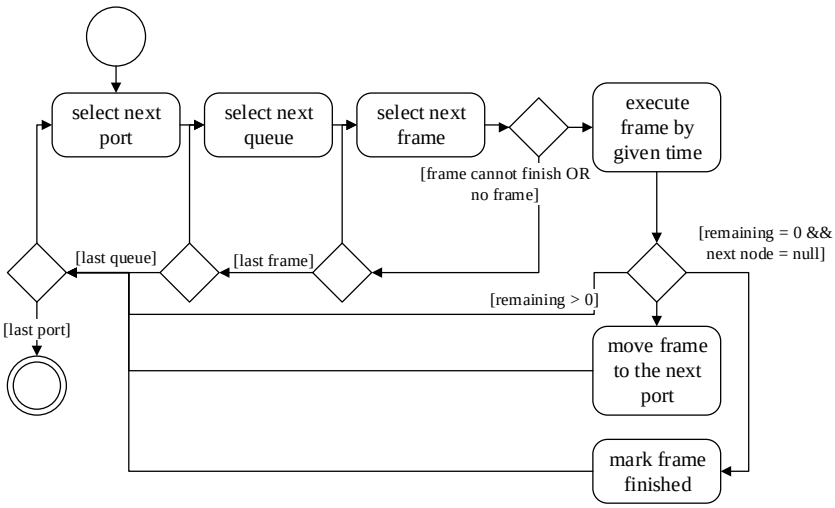


Figure 4.12: Frame execution

The simulator has support also for IEEE 802.1Qav (AVB credit-based shaper) and IEEE 802.1Qbu (preemption) in addition to IEEE 802.1Qbv (TSN time-aware shaper). All the features can be used alone or combined. The activity diagrams for credit-based shaper and preemption are shown in figures 4.13 and 4.14 respectively.

Both activity diagrams have very similar characteristics compared to the "normal" frame execution. The combinations of different features are easy to implement – the differences are mainly in the checking if frame can be executed – and therefore not shown here.

4.4.4 Handling time and state

The network has a globally synchronized time as defined in the list of assumptions. The time is stored and incremented in *GlobalTime* object, which uses singleton-pattern to be globally available and the same always.

As the simulator will be run until it reaches a stable state or the maximum runtime is reached, it is important to define what is the stable state and how it is checked. The check is run every *hyperperiod* of time-critical frames of time after ten first hyperperiods has passed.

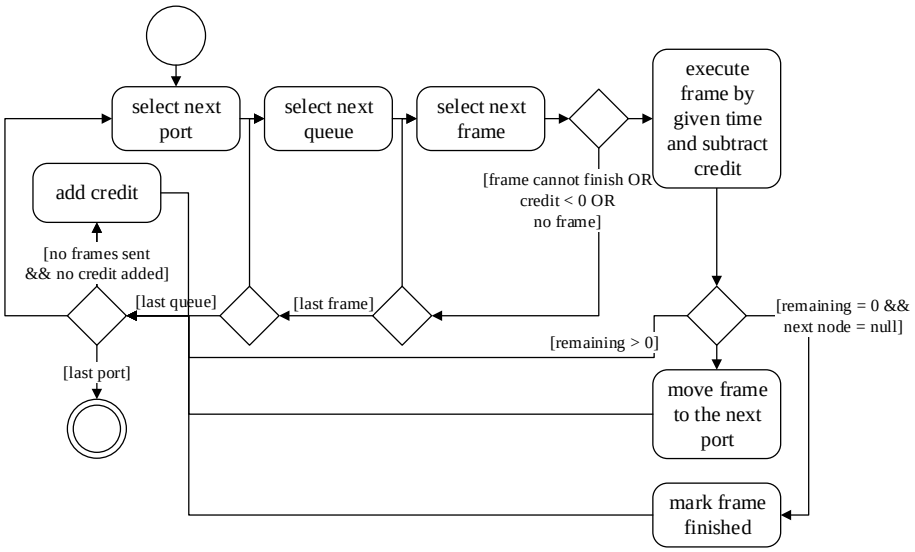


Figure 4.13: Frame execution with CBS

The run time before checks is to gather initial data to compare to. The simulator stores statistics after to a list of statistics after every check. An important detail is that all the entries in the statistics list contain information about *all* the frames transmitted so far and not only between the checks. The activity diagram is shown in the figure 4.15.

The variables *stable*, *reallyStable*, *maxDiff* and *stats* are attributes in the *GlobalScheduler* object and therefore keep their states between checks. Average values are compared for each message ids.

4.5 Summary

Even though the simulator is simply transmitting frames from one node to another, it becomes fairly complex in code. There are also several design choices to be made in order to either keep the simulator small and simple or more customizable. The end result is fairly simple, but it is not very complex to customize. However, since the simulator was modified from an existing one, it possibly has some features done ineffectively.

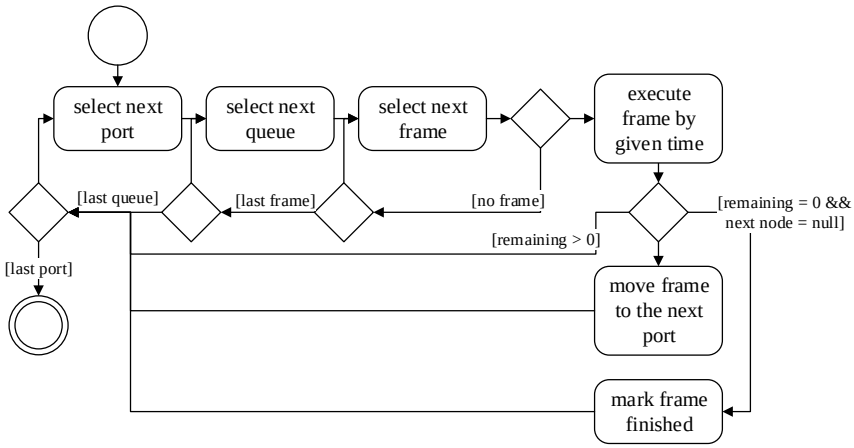


Figure 4.14: Frame execution with preemption

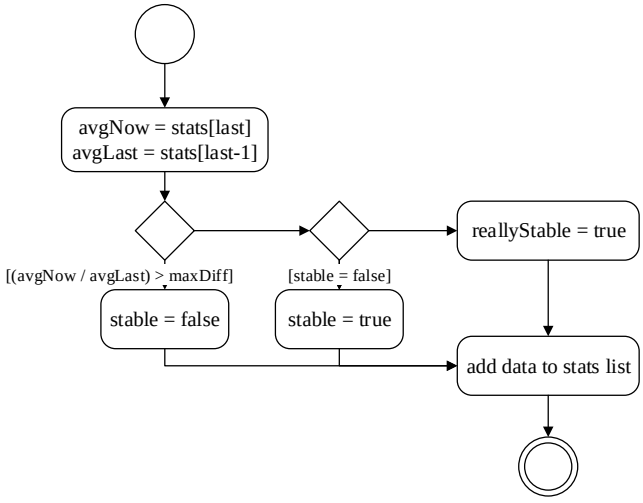


Figure 4.15: Checking stable state

Chapter 5

Evaluation

This chapter discusses about verifying the simulator really works and the results.

5.1 Testing the simulator

The simulator has to be tested with known input and output to verify it works as designed. For testing the simulator, randomness was removed in order to get results to be compared with an example. A test run is shown in figure 5.1.

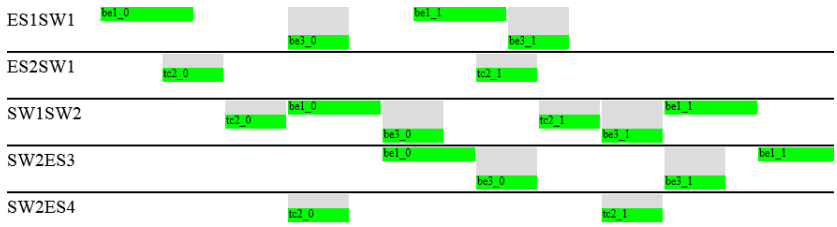


Figure 5.1: Data flow for test run

The test run gives identical results as seen in figure 2.10. It proves the simulator

works as planned and protected windows actually give zero extra latencies for time-critical frames.

Table 5.1: Output data for test run

Frame	Best-case delay	Average delay	Worst-case delay
be3	227840	227840	227840
tc2	170880	170880	170880
be1	341760	370240	398720

Further analysis of the output data from table 5.1 verify the simple example gives excepted results. The results are correct also with longer runs with randomness.

5.2 Evaluation and results

Chapter 6

Conclusion

This chapter discusses the results and observation of the evaluations

6.1 Future work

Future work

Appendix A

Stuff

This appendix is full of stuff ...

Bibliography

[1] G. Arlen. The internet of things. *Multichannel News*, 35(8):10 – 12, 2014.

[2] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-event system simulation*. Pearson, 2010.

[3] J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. Scoredos, P. Stanfill, D. Stuart, and R. Urzi. A research agenda for mixed-criticality systems. Technical report, IEEE Real-Time and Embedded Technology and Applications Symposium, April 2009.

[4] L. G. Birta and G. Arbez. *Modelling and simulation*. 2013.

[5] K. Chan, B. Bensaou, and D. Tsang. Credit-based fair queueing (cbfq). *ELECTRONICS LETTERS*, 33(7):584–585, 1997.

[6] D. Fedyk and D. Allan. Ethernet data plane evolution for provider networks [next-generation carrier ethernet transport technologies]. *Communications Magazine, IEEE*, 46(3):84–89, March 2008.

[7] M. Fiedler, T. Hossfeld, and P. Tran-Gia. A generic quantitative relationship between quality of experience and quality of service. *IEEE NETWORK*, 24(2):36–41, 2010.

[8] N. Finn. Detnet problem statement. `http://www.ietf.org/proceedings/91/slides/slides-91-detnet-9.pptx`.

[9] IEEE. Ieee 802.1 av bridging task group. `http://ieee802.org/1/pages/avbridges.html`.

[10] IEEE. Ieee 802.1q-2014. `http://www.ieee802.org/1/pages/802.1Q-2014.html`.

[11] IEEE. Ieee std 802.1q^{TM}-2005. Standard, Institute of Electrical and Electronics Engineers, Inc, 05 2006.

[12] IEEE. Ieee p802.1qbv/d2.1. Standard (draft), Institute of Electrical and Electronics Engineers, Inc, 10 2014.

[13] J. Kleijen. Validation of models: statistical techniques and data availability. In *Simulation Conference Proceedings, 1999 Winter*, volume 1, pages 647–654. IEEE, 1999.

[14] H. Kopetz. *Real-time systems : design principles for distributed embedded applications*. Springer Verlag, 2011.

[15] H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, 2003.

[16] H. Kopetz and G. Grunsteidl. Ttp-a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, 1994.

[17] A. Law and W. Kelton. *Simulation modeling and analysis*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 2000.

[18] A. M. Law. How to build valid and credible simulation models. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 24–33. IEEE, 2009.

[19] N. Maalel, E. Natalizio, A. Bouabdallah, P. Roux, and M. Kellil. Reliability for emergency applications in internet of things. In *Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on*, pages 361–366. IEEE, 2013.

[20] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt. Extending ieee 802.1 avb with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic. In *VNC*, pages 47–54, 2013.

[21] D. Pannell. Audio video bridging gen 2 assumptions, 2012.

[22] I. Petrila, V. Manta, and F. Ungureanu. Uniformity and correlation test parameters for random numbers generators. pages 231–236, 2014.

[23] F. Redmill. Understanding safety integrity levels. *MEASUREMENT and CONTROL*, 32(7):197–200, 1999.

[24] T. Skeie, S. Johannessen, and O. Holmeide. The road to an end-to-end deterministic ethernet. In *Factory Communication Systems, 2002. 4th IEEE International Workshop on*, pages 3–9. IEEE, 2002.

[25] B. Stafford. Ieee 802.1q header. `http://commons.wikimedia.org/wiki/File:Ethernet_802.1Q_Insert.svg`.

[26] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz. Tomorrow's in-car interconnect? a competitive evaluation of ieee 802.1 avb and time-triggered ethernet (as6802). In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5. IEEE, 2012.

[27] D. Tamas-Selicean, P. Pop, and W. Steiner. Synthesis of communication schedules for ttethernet-based mixed-criticality systems. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 473–482. ACM, 2012.

[28] P. Thaler, N. Finn, D. Fedyk, G. Parsons, and E. Gray. Media access control bridges and virtual bridged local area networks. `https://www.ietf.org/meeting/86/tutorials/86-IEEE-8021-Thaler.pdf`.

[29] A. Zafirow. Modeling and simulation of the ttethernet communication protocol. Master's thesis, Technical University of Denmark, 2013.