Providing a Solution for Configuration of Linux end-hosts in Time-Sensitive Networks

Lösning för konfiguration av Linux-end-hosts i Time-Sensitive Networks

Jesper Olsson
Nils Alonso

Department of Mathematics and Computer Science

Computer Science

C-dissertation 15hp

Supervisor: Hamza Chahed, Andreas Kassler

Examiner: Leonardo Martucci

Date: 2023-05-31

Acknowledgements

Thanks to our supervisors Hamza Chahed and Andreas Kassler for giving us the opportunity to do our thesis and providing guidance, to Héctor Blanco Alcaine from Intel for providing extensive help regarding DETD during the entire project, and to Tobias Vehkajärvi for helping us set up the hardware.

Abstract

Time-critical networks of various types are widely used in fields such as industrial automation. Many of these time-critical networking solutions are proprietary and closed, which can make them costly to work with. An alternative to these legacy solutions is Time-Sensitive Networking. Time Sensitive Networking, or TSN, is an open standard for time-critical communication over Ethernet hardware and protocols. Compared to proprietary and closed legacy solutions, a TSN can be easier to set up. There is still however a challenge in configuring a TSN since the configuration process is hardware dependent. This thesis sets out to ease the configuration process, making it more user-friendly by providing a tool for the generation of end-host configurations. Currently, no such readily available tool exists for configuration of Linux end-hosts in TSNs. This is done by implementing extensions to the incomplete TSN configuration middleware DETD to a state where it is a suitable solution to this problem. The extensions made to DETD consist of implementing support for configuring listener streams, adding the ability to configure the TAPRIO queueing discipline, and adding support for an additional network interface card in the form of the Intel I210. To verify the functionality of these extensions a simple testbed using two real-time Linux machines is used.

Contents

Acknowledgements **i**

Abstract **iii**

Figures **ix**

1 Introduction **1**

1.1 Background 2

1.2 Problem Statement 3

1.3 Objective and Goals 4

1.4 Ethical Considerations 4

1.5 Method 4

1.6 Stakeholders 5

1.7 Division of Labor 5

1.8 Delimitations 6

1.9 Disposition 6

2 Background **9**

2.1 Time-Sensitive Networking 9

2.2 IEEE 802.1Qbv Time Aware Shaper 11

2.3 Qdiscs 11

2.3.1 Time Aware Priority Qdisc 12
2.3.2 Earliest TxTime First 13
2.4 Time Synchronization Using PTP 13
2.5 DETD 14
2.5.1 Proxy 16
2.5.2 Service 16
2.5.3 Manager 17
2.5.4 InterfaceManager 17
2.5.5 Mapping 17
2.5.6 Scheduler 17
2.5.7 Interface 18
2.5.8 Device 18
2.5.9 SystemConfigurator 19
2.5.10 DeviceConfigurator 19
2.5.11 QdiscConfigurator 19
2.5.12 VlanConfigurator 19
2.5.13 Ethtool 19
2.5.14 Tc 20
2.5.15 Ip 20
2.5.16 Explanation of add talker function in DETD 20
2.6 Traffic Generation using Txrx-tsn 21

3 Design **23**
3.1 Add Listener 23
3.2 Added support for TAPRIO Modes 24
3.3 Options for selection and configuration of TAPRIO modes 24
3.4 I210 Support 25
3.5 Conclusion 26

4 Implementation **27**

4.1 Testbed 27

4.1.1 Intel I225 Machines and I210 Machine 27

4.2 Linux Distribution and Kernel 28

4.3 Extensions to DETD 29

4.3.1 Add Listener 29

4.3.2 Support for TAPRIO modes 31

4.3.3 Options for selection and configuration of TAPRIO modes . . . 34

4.3.4 I210 support 34

4.4 Extending and using Traffic Generation Software Txrx-tsn 35

5 Results **37**

5.1 Result Overview 37

5.2 Extensions to DETD 38

5.2.1 Add Listener 38

5.2.2 Taprio Modes and Options 39

5.2.3 I210 Support 39

6 Conclusions **41**

6.1 Discussion 41

6.1.1 Problems 42

6.2 Conclusions 42

6.3 Future Work 43

6.3.1 Integration of DETD in a larger TSN 43

6.3.2 Supporting More Devices 43

6.3.3 Integrating Time-Synchronization 43

6.3.4 Diagnostics Mode 44

Bibliography **45**

Attachments **48**

A Abbreviations **51**

List of Figures

2.1 Chart of fully centralized TSN 10
2.2 Visualization of TAPRIO modes 12
2.3 Chart of DETD configured TSN structure 14
2.4 DETD internal class structure 16
2.5 General workflow of DETD running add talker[1] 20

3.1 Workflow chart of add listener 24

4.1 Structure of the testbed 28

Chapter 1

Introduction

The industrial automation industry has long used various real-time Ethernet solutions for its deterministic communications needs and has recently started adopting time-sensitive network (TSN) solutions [2]. One reason for this is the openness and adaptivity of TSN networks compared to legacy solutions. TSN is an extension to Ethernet technology enabling it to handle real-time traffic as well. However, by being standardized, it solves the problem of incompatibility that arises when using hardware from different vendors.

For a time-sensitive network using time-aware scheduling to function, devices need to follow a schedule that is pre-calculated and pre-configured in all the network devices. These devices also need to have their hardware clocks synchronized for their schedules to work correctly. Automating configuration can be challenging, especially in a Linux environment, which is why non platform-specific software to automate this can be useful.

DETD is a prototype of a middleware used to configure Linux-based end-hosts of time-sensitive networks. The purpose of this software is to make an interface between time-sensitive applications and the underlying system to make configuration of time-sensitive networks easier. However, DETD in its original state lacks vital functionality.

This thesis aims at advancing the state of the Linux TSN environment by enabling

the necessary functionalities to configure a TSN end-host using DETD.

1.1 Background

Modern industrial automation utilizes both edge computing[1] and virtualization to reduce the cost of operation, increase efficiency, and increase flexibility. For example, virtualization of Programmable Logic Controllers (PLCs) allows previously physical PLCs to be virtualized. This enables PLCs to be moved to a server that can host a large number of Virtual PLCs (vPLCs) saving costs. Flexibility is also improved by the possibility of performing diagnostics and data analysis in real-time and in parallel to other operations [3].

For this to be feasible in an industrial automation environment, hard real-time communication between devices is crucial. This means that communication needs to be deterministic. Deterministic communication means that the network provides Quality of Service (QoS) guarantees on the forwarding of network traffic.

Although legacy solutions could provide QoS guarantees to real-time traffic they suffer from being proprietary and closed, meaning that there is very little monitoring of these solutions. Because of this, adding another use case to a production line using one of these legacy solutions can be a substantial task that requires the engineering of the entire production line to be done. The proprietary nature of these legacy solutions leads to the added complexity of every type of traffic having a different network infrastructure. Working with these legacy solutions also require specialized knowledge of how they work, adding another step of expense and complication.

IEEE 802.1 time-sensitive networking alleviates these problems. TSN enable industrial automation solutions that are open, distributed and adaptive. This is done by using standardized Ethernet hardware and protocols to construct a real-time network,

[1] computing which is performed by or near the end-device

rather than the proprietary hardware and protocols used in the Legacy solutions. This also enables end-hosts to run non-proprietary operating systems such as Linux. A TSN consists of switches, referred to as TSN-bridges, and end-hosts which have network interfaces connected to the network. In a TSN context, these end-hosts can be configured as talkers and listeners.

IEEE802.1 TSN has a set of mechanisms that guarantee the quality of service (QoS) of communication. A few examples of these mechanisms are time synchronization and time aware scheduling. Without time synchronization, any time-based communication is rendered impossible. To address this, time aware scheduling utilizes a centralized pre-calculated schedule among the switches (TSN-bridges) in the network. This schedule dictates which traffic is forwarded through the network at a particular time.

1.2 Problem Statement

All of the previously mentioned TSN mechanisms require both the switches and the end-hosts to be configured in a specific way for the TSN to function. To configure the switches, a readily available open solution already exists in the form of OpenCNC [4]. OpenCNC enables the calculation and distribution of a centralized schedule among the switches in a TSN. However, no such open and readily available solution exists for the configuration of end-hosts. There is a need for this since configuration of end-hosts in a TSN can be complicated and vendor-specific. Different network interface devices have different capabilities, which leads to the configuration process being different for each device. This adds difficulty to configuring the end-hosts in a TSN.

The aim of this thesis is to ease this difficulty by providing a method for configuring TSN end-hosts that provides abstraction from device specific aspects.

1.3 Objective and Goals

The goal of this project is to ease the configuration of Linux TSN end-hosts by hiding hardware complexity and providing a tool independent of network interface card-vendors for configuring Linux end-hosts in a time-sensitive network.

1.4 Ethical Considerations

This project has no ethical issues or considerations since the work is carried out exclusively on networking hardware and software where no personal or sensitive data is used. Since ethics is related to human involvement, it is hard to encounter any ethical issues in the work related to this thesis. A TSN is a closed network that is not connected to the internet. Because of this, consideration to data leaks and hacking is not necessary. Potential malicious use-cases of TSNs could be considered in some cases, but individual use cases of TSNs are outside the scope of this thesis and is therefore not something that will be considered. Since the use case of a TSN is mainly industrial automation, there can be serious consequences for misconfiguration. DETD is however a prototype and a proof of concept, rather than a commercial product, which means that this does not need to be considered.

1.5 Method

In order to solve the problem of configuring Linux-based TSN end-hosts, DETD was suggested as an open source alternative that simplifies the process of configuring end-hosts in a time-sensitive network. This involves decoupling the configuration process from platform and device specific aspects and providing a layer of abstraction from implementation specific aspects [5]. DETD supports the configuration of multiple network interface cards using the same setup process. DETD also provides some automatic

calculations of certain parameters. It is however a prototype for experimentation and is not intended to be a production-grade software. Since DETD is a software running on the individual end-hosts it can not be used to configure a TSN in its entirety. It can however act as a substantial step in the TSN configuration process.

In its original state DETD lacks functionality that is critical to close the configuration loop of a TSN network. It only allows for the configuration of an end-host as a talker. An end-host being set up as a talker means that the end-host can send packets into the network. Configuration of a listener, which enables the end-host to receive packets from the network, is not yet possible. Without this functionality, DETD can not be used as the only end-host configuration tool in a TSN. Apart from listener support, support for setting different modes of the Linux queueing discipline TAPRIO (Time Aware Priority Shaper) including the options for choosing the specific mode, and support for the Intel I210 NIC are also features that could further augment DETD as a viable option to configure TSN end-hosts. This project sets out to implement these previously mentioned features.

1.6 Stakeholders

The project has two main stakeholders, which are Seco and Intel. Seco has provided the two computers used as end-hosts for the TSN, and they benefit from the proof of concept of how their boards can be used. The other stakeholder is Intel, since extensions to the Intel software DETD are implemented during the project.

1.7 Division of Labor

During the project, the division of labor has been approximately equal. The project was worked on partially together on the same task and partially worked on tasks in

parallel. The first stage of setting up the machines was handled by both authors, but troubleshooting the real-time kernel on the I225 Machines was handled by Nils. Installation of DETD was done by Jesper for the most part. In the DETD software, Jesper implemented support for the Intel I210 network card, extra options for the configuration of TAPRIO, and the two missing TAPRIO modes. Nils implemented adding listener streams. Nils also worked on using and extending Txrx-tsn.

The thesis was written in cooperation, where both authors wrote and reviewed the written material. The main responsibilities were however divided similarly to the project, where each author wrote about the topic they had experience with.

1.8 Delimitations

The first delimitation is that the network used is simply built up of two machines connected to each other. No functionality is tested using any switched network or more than two interfaces at a time. Another delimitation is that the performance of the TSN configurations is not analyzed, since it is not meaningful without an actual switched network. Because of time constraints, another delimitation is that the Intel I210 support is not tested using traffic generation.

1.9 Disposition

This thesis is structured as follows:

- Chapter 2 summarizes the TSN concepts used in the thesis and describes the software DETD in detail.
- Chapter 3 describes the design of the extensions to DETD, and the structure of the testbed.

- Chapter 4 describes the process of implementing the changes made to DETD and setting up the testbed.
- Chapter 5 describes the experiments made to confirm the correctness of what has been implemented in Chapter 4.
- Chapter 6 discusses the results, the work process, and future work that could be made.

Chapter 2

Background

This chapter gives a general explanation of time-sensitive networks and technologies relevant to TSNs. The chapter also contains a general description of DETD and its classes and a short explanation of the traffic generation software Txrx-tsn.

2.1 Time-Sensitive Networking

Time-sensitive networking is a standard for enabling deterministic communication over physical switched Ethernet networks. Determinism in this case means that packets are sent and received at predictable times within a certain margin of error [2]. Although regular Ethernet communication can be very fast, it can not provide this type of determinism.

A TSN consists of a switched network of one or multiple interconnected TSN-bridges and end-hosts that send and receive packets through the network. End-hosts can be set up as talkers, listeners or both. Talkers transmit traffic into the network while listeners receive traffic. For these TSN-bridges and end-hosts to work they need appropriate configuration. There are a few models of how to accomplish this, which are fully centralized TSN, centralized network/distributed user TSN or a fully distributed

TSN.

The fully centralized TSN contains both a CUC (Centralized User Configuration) and a CNC (Centralized Network Configuration) where configuration of the TSN-bridges are handled by the CNC and configuration of the end-hosts are handled by the CUC. A chart of the general fully centralized TSN structure can be seen in figure 2.1. A centralized network/distributed user TSN is structured similarily to a fully centralized TSN apart from it being set up without any CUC. The fully distributed TSN differs the most since it uses a distributed configuration protocol [6].

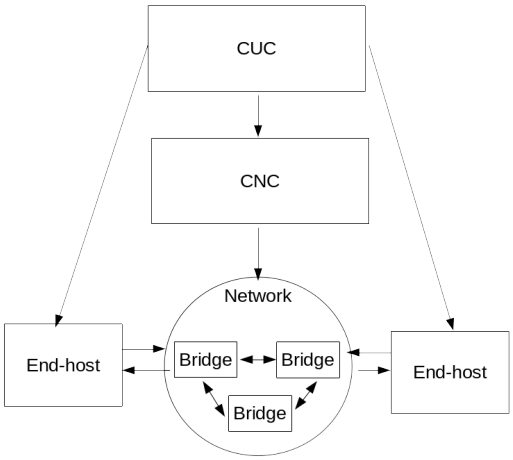

Figure 2.1: Chart of fully centralized TSN

The CUC is a function that communicates with both the CNC and the end-hosts to trigger the configuration of the entire network. The specific functionality of the CUC is however not specified in the IEEE 802.1 standard [6].

The CNC is a function that communicates with all the bridges to calculate a global schedule and distribute it to all the bridges. A CNC can run on a dedicated machine somewhere in the network or on a non-specific machine with another role such as an end-host or a TSN-bridge.

2.2 IEEE 802.1Qbv Time Aware Shaper

IEEE 802.1Qbv time aware shaper is a shaping mechanism working in switches within an IEEE 802.1 time-sensitive network. IEEE 802.1Qbv enables the profiling of traffic by placing a gate at every queue capable of allowing time-critical traffic to be sent while preventing all other traffic for a window of time [7].

This is however not entirely unproblematic. If best effort transmissions are not completed before time-critical traffic of the next cycle starts, time-critical traffic will be disrupted. This is solved by allocating time as a guard band in front of each window of time-critical traffic. This guard band is as large as the transmission length of the largest packet. During the guard band, new transmissions can not start, only transmissions that have already begun can finish transmitting. This is needed to guarantee determinism. This is however not a very effective use of bandwidth [8].

2.3 Qdiscs

Some of the TSN mechanisms, including TAS (Time Aware Shaper), are integrated in the Linux environment using qdiscs. Qdisc is short for queueing discipline, and is a part of traffic control in Linux. Every network interface has a queue that is controlled by a queueing discipline. When a packet is to be sent from the kernel to an interface, it is first enqueued to that device's qdisc. The packets are then sent to the network interface in order of the queue whenever it is possible [9]. The main qdisc used in this thesis is Time Aware Priority Shaper (TAPRIO), which is a Linux implementation of TAS [10]. Another qdisc used is Earliest TxTime First (ETF) qdisc [11].

2.3.1 Time Aware Priority Qdisc

Time Aware Priority Shaper (TAPRIO) is a queueing discipline (qdisc) in Linux Traffic Control (tc). It is based on a simplified version of the scheduling state machine defined by IEEE 802.1Q-2018 [12] Section 8.6.9, which allows for the configuration of a sequence of gate states [10]. The sequence is executed in order for the pre-determined amount of time specified in the entered schedule. The configuration of TAPRIO using tc contains three different modes. These modes are software mode, txtime-assist mode, and full-offload mode. For software mode, the gate control list is executed by the kernel. In txtime-assist mode, TAPRIO sets a transmission timestamp and then utilizes the ETF queuing discipline to sort and transmit the packets at the correct time. For full-offload mode, the gate control list is passed to the device to execute it in hardware [10]. The different TAPRIO modes are visualized in Figure 2.2. Differences in performance exist

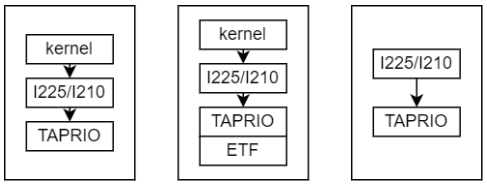

Figure 2.2: Visualization of TAPRIO modes

across the three different modes, which is measured by the jitter[1] of sent packets. The full-offload mode is expected to have the best performance among the three due to the execution in hardware compared to the other two modes being executed by the kernel. Secondary in performance is expected to be txtime-assist mode due to the scheduling of packets ahead of their transmission.

[1] Variation of latency

2.3.2 Earliest TxTime First

Earliest TxTime First (ETF) implements functionality commonly known as LaunchTime and is installed under another qdisc. LaunchTime entails that the qdisc controls when a packet should be dequeued from traffic control to the network interface card and if offload is supported by the network interface card it may be used to control when packets are sent from the card. This is achieved by buffering the packets until a configured time before their transmission time. ETF also ensures packets are sent in order of earliest transmission time first and if packets have a transmission time in the past or expire while waiting to be dequeued, the packets are instead dropped [11].

2.4 Time Synchronization Using PTP

Another mechanism of an IEEE 801.2 TSN is clock synchronization, which can be crucial for scheduled traffic. Any time-based schedule in the network will be rendered useless if devices using the schedule do not follow the same clock. In a TSN, the local clocks of the end-hosts in the network are synchronized using the PTP protocol. In PTP, one host is selected to be a what is referred to as a grandmaster. The role of the grandmaster is to use its local clock as a reference to set the remaining local clocks. The most suitable grandmaster is selected using the best master algorithm, and the hierarchy is organized in a synchronization spanning tree with the grandmaster as the root. Ports in the network can be set as master, slave, passive, or disabled entirely. To synchronize the clocks, the root (grandmaster) sends a synchronization message that propagates through the synchronization spanning tree. This packet contains a timestamp. For an end-host to calculate the correct time, the reception timestamp is subtracted from the transmission timestamp to calculate what is called the residence time, which is the correct time for that particular device [2].

2.5 DETD

DETD was initially developed by Intel, and is a software used for configuring the network interface cards with a queueing discipline using tc, controlling the network driver and hardware settings using ethtool, and configuring VLAN using ip link [5]. DETD's role in the end-hosts of a TSN is shown in figure 2.3.

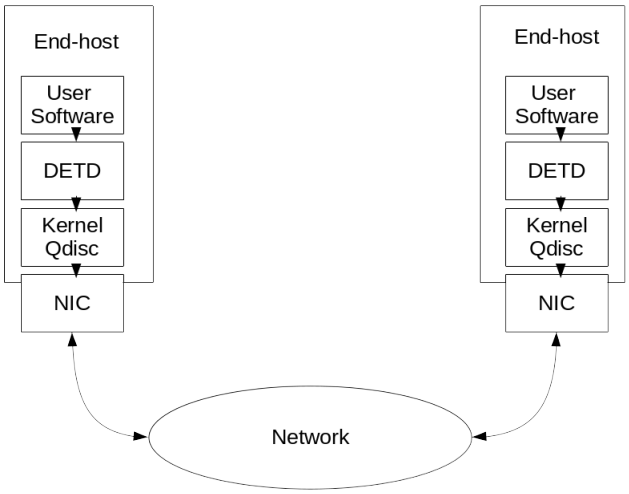

Figure 2.3: Chart of DETD configured TSN structure

Initially, DETD's supported devices were limited to Intel I225-LM, Intel I225-IT, and Intel Atom x6000E Series integrated TSN controller. The device is identified through a list of PCI IDs for the device in question [5].

To set up a stream with a particular device, the user provides information such as interface, transmission cycle, size of packets, and transmission time offset, among others. This is then used to configure the stream and traffic. Some of this information is also used to determine if conflicts with existing streams exist.

The connection from user application to DETD is made through Protobuf, which is a platform-neutral language developed by Google for serializing structured data. It is designed to be more lightweight and faster than other options, such as XML, and

is mostly used for defining communication protocols and for data storage [13]. Since DETD uses Protobuf to pass configuration data, applications using DETD can be written in any language that has an available compiler for Protobuf.

When DETD receives the information provided by the user, this information is used to configure the network interface card. The configuration can be split into three distinct operations: device configuration, queueing discipline configuration, and VLAN configuration.

Device configuration consists of handling hardware and driver-related settings for the network interface card. These settings are important for the performance or function of the time-sensitive network. For performance, Energy Efficient Ethernet is disabled. This is due to the fact that the Ethernet link will go into sleep mode for short periods of time and will need to be woken up, which adds delays that may ruin timing requirements for an application.

The queueing discipline configuration is the configuration of the TAPRIO queueing discipline. It is responsible for applying the correct settings depending on the capabilities of the underlying device. The capabilities may decide what TAPRIO mode the network interface card uses for configuration.

The VLAN configuration handles the mapping of the Linux internal packet priority to the VLAN header priority field for outgoing frames [14].

DETD has a modular structure consisting of a few classes that handle different parts of the configuration process. The classes are described in detail in the following sections. The structure of DETD internals in the context of configuring a talker is described in figure 2.4.

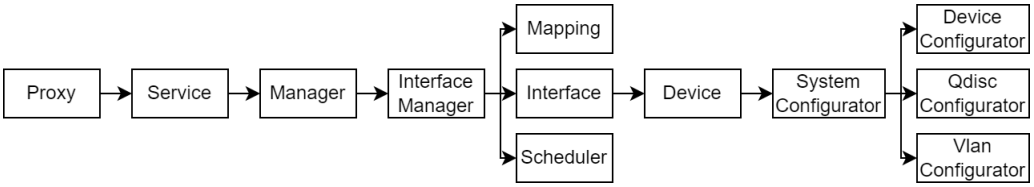

Figure 2.4: DETD internal class structure

2.5.1 Proxy

The Proxy class is a placeholder for a user application. It is responsible for the connection to the Service class using a Unix socket and passing the information required to configure a TSN talker end-host. There are methods and functions for sending and receiving data through sockets and functions for passing information to the Service class for the configuration of end-hosts using Protobuf.

2.5.2 Service

The Service class is responsible for handling the requests sent from the application utilizing DETD. It receives the data sent through Protobuf, parses it, and calls the Manager class with the data packaged into an object. Once the device is configured, it is also responsible for passing data back to the caller, such as the configured VLAN interface and socket priority to the socket dedicated to real-time traffic. In this case, the system service receives the data passed from Proxy. The class contains functions for receiving data from Protobuf and parsing it into a structured format for the later configuration of the device. It also has the function for starting the configuration of said device.

2.5.3 Manager

The Manager class is responsible for keeping a list of talkers and will attempt to further the add talker request to InterfaceManager if none are already configured on that device. It also returns the return values from InterfaceManager, which it calls to begin the configuration.

2.5.4 InterfaceManager

The InterfaceManager class is responsible for configuring the device. It achieves this by calling further classes to get device information, adding traffic to the schedule, handling socket priority mapping, ensuring the device can implement the schedule, and configuring the device itself. After configuring the device, it returns the VLAN interface and socket priority.

2.5.5 Mapping

The Mapping class handles the mapping and allocation/deallocation of resources, such as keeping track of traffic classes, which resources are allocated to streams, and which hardware queue corresponds to which traffic class. It also handles operations such as mapping socket priority to traffic class. It contains functions for mapping Linux internal packet priority to VLAN header PCP field for outgoing traffic, assigning socket priority to traffic classes, assigning socket priority to stream, assigning traffic class to stream, assigning queue to stream, and clearing assigned resources.

2.5.6 Scheduler

The Scheduler module contains a number of classes tasked with handling scheduling and traffic. These classes are Scheduler, Schedule, TrafficSpecification, Traffic, Slot, Configuration, and StreamConfiguration.

Scheduler is a class that deals with storing and managing the traffic schedule. It stores both best effort and scheduled traffic. For managing the schedule, it uses the Schedule class, which is a class inheriting from List. This Schedule class holds a list of slots defined in the Slot class and is also responsible for ensuring there is no conflict between traffic as well as managing the content of its own list. The Slot class it utilizes is an object for the storage of data such as traffic size and format.

The Traffic class contains storage for all necessary information about the traffic used by DETD. This includes many of the values provided by the user as well as some additional calculated values.

The TrafficSpecification class stores the values related to traffic and the StreamConfiguration class stores the values related to the configuration of the stream. Both TrafficSpecification and StreamConfiguration contain checks to ensure the values provided to them has the correct format. The Configuration class is the object for storing the Interface, TrafficSpecification, and StreamConfiguration objects. The configuration object is used to pass the user input through the application.

2.5.7 Interface

This class stores driver information fetched using ethtool, such as PCI ID. It also provides a way to get the link rate and furthers the device configuration request.

2.5.8 Device

The Device class is an abstract class for implementing devices. It is used to identify the device by PCI ID, stores information about the device, such as the number of transmission and receive queues, and what features the device supports. It has functions defined for configuring the end-host, getting link rate, and verifying that the device supports a certain feature.

2.5.9 SystemConfigurator

The SystemConfigurator class keeps track of VLAN ids that are already configured and is responsible for calling lower-level classes for the configuration of the device.

2.5.10 DeviceConfigurator

The DeviceConfigurator class is responsible for configuring the device hardware settings. This is accomplished by utilizing ethtool through the ethtool class and is used for disabling Energy Efficient Ethernet (EEE), among other things.

2.5.11 QdiscConfigurator

The QdiscConfigurator class is responsible for configuring the TAPRIO queueing discipline. This is accomplished using the tc command through the Tc module and handles the selection of TAPRIO mode.

2.5.12 VlanConfigurator

The VlanConfigurator class is responsible for setting the Linux internal packet priority to VLAN header PCP field for outgoing traffic. This is accomplished using the ip system command through the Ip class.

2.5.13 Ethtool

The Ethtool module provides classes for calling ethtool to fetch information about the device as well as configuring the device's hardware settings. Ethtool is a Linux command used for querying and controlling drivers and hardware settings for network devices [15].

2.5.14 Tc

The Tc module provides classes to execute iproute2's tc command for the configuration of queueing disciplines. It calls the tc system command with parameters provided by the user or calculated by DETD. Tc is a Linux system command which is used to configure traffic control in the Linux kernel [9].

2.5.15 Ip

The Ip module provides classes to execute iproute2's ip command for the configuration of the VLAN link using parameters provided by the user or calculated by DETD. Ip is the Linux command used to show or manipulate routing, network devices, interfaces and tunnels. [16]

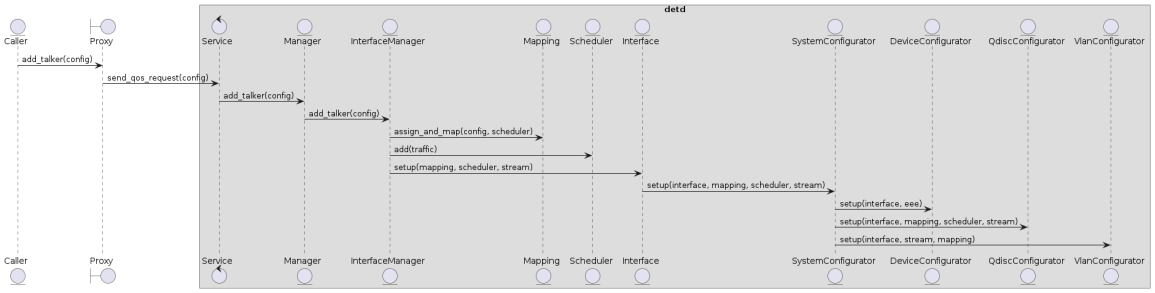

Figure 2.5: General workflow of DETD running add talker[1]

2.5.16 Explanation of add talker function in DETD

As seen in Figure 2.5, the general flow of adding a talker stream with DETD consists of passing a configuration object from the user application to the system service. The add talker request is passed to Manager and InterfaceManager, which in turn calls on the Mapping and Schedule modules to Allocate resources and ensure traffic is schedulable. It also calls SystemConfigurator, which distributes the workload of configuring the

talker stream and device to the DeviceConfigurator, QdiscConfigurator, and VlanConfigurator classes.

2.6 Traffic Generation using Txrx-tsn

Txrx-tsn is a software developed by Intel with the purpose of testing time-sensitive networks. Its use is in testing low-latency transmission and reception through generating and receiving traffic. The transmission and reception is timestamped to showcase the latency of each packet being sent [17]. Txrx-tsn provides both user timestamps, meaning timestamps from the operating system clock, and hardware timestamps, which are taken from the network interface's hardware clock. Packets can be sent using either AF_PACKET or AF_XDP, which are two different Linux socket interfaces [17].

Chapter 3

Design

This chapter describes the design of the extensions to the DETD software.

3.1 Add Listener

In its original state DETD lacks the functionality to configure a device as a listener, which is a necessary function for a TSN. As such this function needed to be implemented as a part of the project. It was also necessary to implement this functionality using the same classes and a similar workflow to not break the original design of DETD.

For the add listener function to work it needs to subscribe to the multicast MAC address that the talker uses to send packets, handle vlan mapping and handle device tuning through the Linux command ethtool. Multicast MAC has been added as a parameter in the config file used when calling the add_listener function. This config file is otherwise identical to the one used when adding a talker. Vlan mapping in this case consists of mapping the vlan prio from a packet's header to the packet priority on incoming frames in Linux [14]. In this case this is mapped in a 1:1 fashion. This is implemented using the classes already present in DETD according to the workflow shown in figure 3.1.

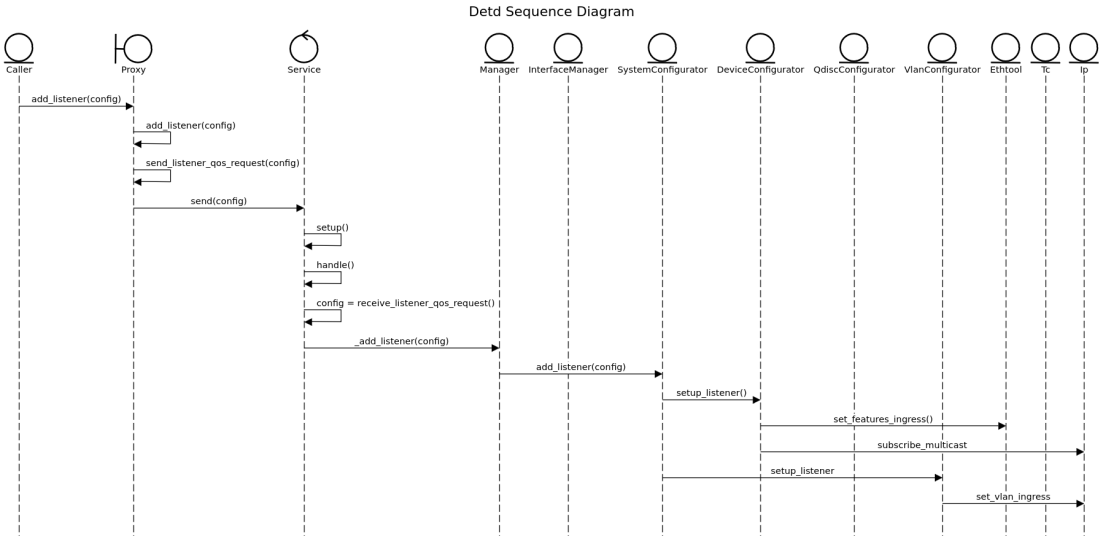

Figure 3.1: Workflow chart of add listener

3.2 Added support for TAPRIO Modes

Initially, only one configuration of TAPRIO was supported, which was the full-offload mode. To provide TAPRIO configurations where the device in use does not support full-offload, or a mode other than full-offload is preferred, it is crucial to have support for the two missing modes. Our solution for this is to provide a way to configure both software mode and txtime-assist mode for TAPRIO through execution of the Linux system command tc.

3.3 Options for selection and configuration of TAPRIO modes

Another issue related to the configuration of the device is the lack of ability to choose what TAPRIO mode to use, which means that DETD will decide the TAPRIO mode with the best performance for the device configuration, which is in the order of first

full-offload and secondarily txtime-assist as described in Section 2.3.1. This is not always a desired behavior due to the different TAPRIO modes having different benefits. Additionally, there was no ability to customize the mapping of traffic class to socket priority. This means that DETD will provide a default static mapping to the TAPRIO qdisc. For some applications, this mapping may not be acceptable.

To solve both of these issues, a similar approach is used. First, an object is created to store optional parameters such as the TAPRIO mode and mapping. This object is then passed along with the other user-application input to the appropriate class, where the options are handled differently. When it comes to the TAPRIO mode option, it is used to select which of the TAPRIO modes to force the device to use. If the device does not support the TAPRIO mode provided, the TAPRIO configuration will instead default to software mode. If no mode is provided DETD will choose the mode with the best performance that the device supports. Regarding the mapping, it is instead used to overwrite the default mapping generation of DETD, which is later applied to the TAPRIO configuration.

3.4 I210 Support

One of the goals of DETD is to provide a less device-dependent way of configuring end-hosts in a TSN. Currently, DETD has a relatively small list of supported devices. It would be useful for DETD to support more devices than it does in its original state, such as the I210 NIC.

To implement support for a network device in DETD, information that identifies the device needs to be provided. Information about the specifications of the device such as hardware queues and hardware settings to be turned on or off is also necessary to provide.

3.5 Conclusion

The extensions made to DETD presented in this chapter make DETD a viable solution for configuring Linux end-hosts in a TSN. In the most basic sense, implementing listener streams in DETD enables its use as a solution for configuring Linux end-hosts in a TSN. Allowing users to set different TAPRIO modes enables a level of configurability that allows DETD to configure a wider range of production lines. Adding Intel I210 support to DETD extends device independence, which contributes to further device abstraction.

Chapter 4

Implementation

This section describes the process of setting up the machines for the testbed and implementing the extensions to the DETD software. It also describes setting up the traffic generation software txrx-tsn.

4.1 Testbed

The hardware testbed needed to be set up with an operating system compatible with DETD and capable of running TSN applications. For this to be possible, a real-time kernel is needed. Since the boards were provided for TSN use, no consideration for what hardware to use was necessary. Figure 4.1 shows the structure of the testbed. The connection to the .60 network is used for remote access and downloading software.

4.1.1 Intel I225 Machines and I210 Machine

For the testbed, two computers built for real-time use were deployed. The model names of these computers are Seco EC77-7000-1124-I1_50 and Seco EC77-7000-1124-I1_50. These computers were both equipped with Intel I225 network interface cards, which have two interfaces each. On each machine, one interface is dedicated to TSN use, while

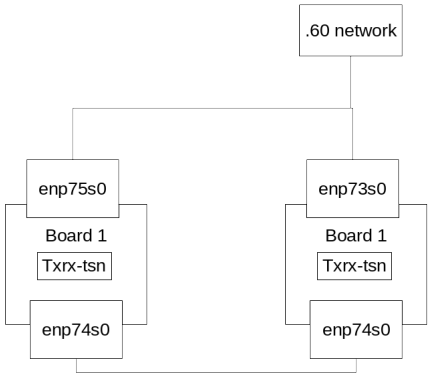

Figure 4.1: Structure of the testbed

the other interface is used for connecting to the machines over ssh and downloading necessary software.

Additionally, a machine with two Intel I210 network interface cards was set up. One of the i210 cards was dedicated to TSN use.

The network consists of a direct connection between the two I225 machines using a single Ethernet cable, while the I210 machine is not connected to any other machine in any of the final experiments.

4.2 Linux Distribution and Kernel

The two end-host machines needed for the experiment both run Debian 12 Bookworm Alpha 2[18] for their operating system. Running a Linux system in a TSN also requires real-time functionality from the operating system. As such, a Linux Kernel patched with RTPatch[19] is used to convert the Linux system into a real-time system. The particular kernel version used is Linux 6.1.12. This setup is also used for the other end-host machine with 2 Intel I210 network cards.

4.3 Extensions to DETD

4.3.1 Add Listener

To be able to test the network using TSN functionality, it was first necessary to implement the support for configuring listener streams.

The function is first called through Proxy with a ListenerConfiguration object as a parameter. This configuration object contains the interface name, interval, packet size, time offset, MAC address, VLAN id, PCP, and the multicast MAC address for the listener to subscribe to. Some of these parameters are inserted into a StreamConfiguration object and a TrafficSpecification object. These are then included in the ListenerConfiguration object. In DETD's current form, this setup is created and passed to the function in the Python script shown in Listing 1

Proxy then calls the Service class. At this point, the necessary system information has been gathered, and is passed to Manager. Since no socket priority needs to be mapped and no schedule needs to be handled, the configuration is passed through InterfaceManager and SystemConfigurator. SystemConfigurator then calls functions in DeviceConfigurator and VlanConfigurator. DeviceConfigurator runs the function set_features_ingress in Ethtool and subscribe_muticast in Ip. Ethtool's set_features_ingress runs the Linux ethtool command

```
ethtool --features <INTERFACE> rxvlan off hw-tc-offload on
```

, which applies device tuning necessary for the interface to act as a TSN receiver. Ip's subscribe_multicast runs the command

```
ip maddr add <STREAM DMAC> <INTERFACE>
```

, which adds the provided multicast MAC address to the list of subscribed MAC addresses for the specified interface. The provided multicast MAC needs to be of a correct format, which in IPv6 means a MAC address between 33:33:00:00:00:00 and

```
from detd import *

def setup_stream_config():

    interface_name = "enp173s0"
    interval = 20 * 1000 * 1000 # ns
    size = 1522                 # Bytes

    txoffset = 0       # ns
    addr = "00:c0:08:a2:d5:73"
    vid = 2
    pcp = 6
    maddress = "33:33:00:00:00:ff"
    interface = Interface(interface_name)
    stream = StreamConfiguration(addr, vid, pcp, txoffset)
    traffic = TrafficSpecification(interval, size)

    config = ListenerConfiguration(interface, stream, traffic, maddress)

    return config

proxy = ServiceProxy()

config = setup_stream_config()
response = proxy.add_listener(config)

print(response)
```

Listing 1: Python script for creating config and running add_listener

33:33:ff:ff:ff:ff. The 33:33 signifies an unreserved multicast MAC[20]. VlanConfigurator calls the function set_vlan_ingress in Ip, which applies VLAN mapping. In this case, vlan priority and packet priority in a 1:1 fashion. This is done through the command shown in Listing 2.

```
ip link add
link     <interface>
name     <new interface name>
type     vlan
protocol 802.1Q
id       3
ingress   0:0 1:1 2:2 3:3 4:4 5:5 6:6 7:7
```

Listing 2: Ip link add Linux command for add listener

4.3.2 Support for TAPRIO modes

Support for software and txtime-assist TAPRIO modes is achieved through the use of a string of the command being sent to the system to be executed. The formatting of the traffic control command provided to DETD is determined through the use of a class with a template, in which the placeholders are replaced with variables provided through user-application input and information calculated by DETD during the instantiation of the class and returns it as a string which is then executed.

When executed, the TAPRIO mode is set by the Linux system command tc qdisc replace. The tc commands for setting the different TAPRIO modes are very similar to each other, with some small differences. Listing 3 is an example of the tc qdisc command for setting up TAPRIO software mode. The dev parameter specifies which interface of a device where the qdisc is to be applied. Parent specifies where the qdisc is to be attached, root being the root of the device. Taprio specifies the qdisc to be used as the TAPRIO qdisc. Num_tc specifies how many traffic classes to use. Map specifies the mapping of traffic class to socket priority. Queues specify the number of queues

```
tc qdisc replace
       dev       <interface name>
       parent    root
       taprio
       num_tc    <number of traffic classes>
       map       <mapping of traffic class to socket priority>
       queues    <hardware queues>
       base-time <base time>
       <schedule entries>
       flags     0x0
       clockid CLOCK_TAI
```

Listing 3: Example of template for setting up TAPRIO software mode

assigned to each traffic class and what range they cover based on an offset. Base-time specifies the UNIX timestamp in nanoseconds when the qdisc is going to be applied. Flags specify a bitmask to choose what TAPRIO mode to use. Clockid specifies the clock to be used by the internal timer of the qdisc.

Txtime-assist mode is set up in a similar way to software mode, but instead with a slightly different template which can be seen in Listing 4. What differs is the addition of the handle and txtime-delay parameters. Handle specifies an ID for the qdisc, which can later be used to specify the qdisc as parent. Txtime-delay specifies the maximum time it may take for a packet to reach the interface from the qdisc.

However, the largest difference is that an ETF qdisc is installed on one of the queues of the TAPRIO qdisc, which enables the functionality of the ETF qdisc on the specific queue it is installed on. Installing the ETF qdisc is different from installing the TAPRIO qdisc as can be seen in Listing 5. The Parent parameter differs with ETF where it is instead provided the handle and queue of where it is to be attached. Etf specifies the qdisc to be used as the Earliest TxTime First qdisc. Delta specifies in nanoseconds how long ahead of the transmission time of the next packet the ETF qdisc prepares for sending. Offload enables the LaunchTime feature. Skip_sock_check skips the check of a socket being associated with packets.

```
tc qdisc replace
        dev       <interface name>
        parent    root
        handle    <ID of qdisc>
        taprio
        num_tc    <number of traffic classes>
        map       <mapping of traffic class to socket priority>
        queues    <hardware queues>
        base-time <base time>
        <schedule entries>
        flags     0x1
        txtime-delay <maximum time for packet to reach interface>
        clockid CLOCK_TAI
```

Listing 4: Example of template for setting up TAPRIO txtime-assist mode

```
tc qdisc replace
        dev       <interface name>
        parent    <queue to install qdisc on>
        etf
        clockid   CLOCK_TAI
        delta     <delta>
        offload
        skip_sock_check
```

Listing 5: Example of template for installing ETF qdisc

4.3.3 Options for selection and configuration of TAPRIO modes

The options parameters are implemented through an object being created for the storage of the options along with functions to validate the format using regex. The object is passed through DETD by being included in the configuration object. However, to ensure compatibility with configurations not utilizing options, if no options are provided, an empty options object is passed. After this, the options are handled differently. To pass the options to DETD along with the configuration it was required to include the options in the Protobuf file.

To decide the TAPRIO mode through the user-application input, the options object is passed to the QdiscConfigurator class, where it is then decided which TAPRIO mode to configure the device with depending on the TAPRIO mode option provided. If the provided mode is not supported by the device, it will instead default to configuring the device in software mode. The options object is also passed to the Mapping class where the mapping option is extracted. This mapping is then used to overwrite the default mapping in the function responsible for generating the mapping.

4.3.4 I210 support

In DETD, device-specific classes inherit from a general device class. Each individual device supported has a corresponding class that inherits from this general device class. For the I210 class to work it needs to contain information about the device itself. The I210 class should contain the device's amount of transmission and receive queues, features supported by the device which in this case is the ability to use txtime-assist mode for TAPRIO, PCI IDs for identification of the device, what hardware settings to turn on and off for the device, and modifying an inherited function used to set device constraints to signify no constraints. The PCI IDs for the I210 card were already specified in the pre-existing template of its class.

4.4 Extending and using Traffic Generation Software Txrx-tsn

Txrx-tsn in its original state lacked the capability to set socket priority independent of VLAN priority. Instead when the VLAN priority was set, socket priority was automatically set to the same value. To enhance it, we implemented a separate socket priority flag. This flag works through overwriting the socket priority set by the VLAN priority flag, and if VLAN priority is not previously set, it leaves VLAN priority as its default value. If AF_XDP is specified, any use of the new flag will return an error message. The choice to use AF_PACKET was made since AF_XDP does not work with the specific kernel used.

Txrx-tsn is in this case used to send packets using AF_PACKET over a multicast MAC using the command in Listing 6. The interface flag specifies which network interface is used, afpkt selects AF_PACKET to be used instead of AF_XDP, transmit is used to send traffic, dst-mac-address is used to specify the mac address to send traffic to, verbose enables the displaying of both hardware and user timestamps for each packet, vlan-prio sets the vlan priority and socket-prio sets the socket priority.

```
./txrx-tsn --interface enp174s0 --afpkt --transmit
--dst-mac-addr 33:33:00:00:00:ff
--verbose --vlan-prio 1 --socket-prio 3
```

Listing 6: Txrx-tsn sender command

The listener is set up using the command in Listing 7. The flags described in Listing 6 also apply here. The receive flag sets up txrx-tsn to receive traffic.

```
./txrx-tsn --interface enp174s0 --afpkt --receive --verbose
```

Listing 7: Txrx-tsn receiver command

Chapter 5

Results

In this chapter, the implementation of extensions to DETD are evaluated. The experiments are described in detail and the results for said experiments are presented.

5.1 Result Overview

The result of this project is a solution for generating configurations for Linux end-hosts in time-sensitive networks. This is presented in the form of an extended version of the software DETD. This extended version of DETD is used on individual Linux end-hosts to apply listener or talker configurations. This version of DETD supports a set of Intel devices, which are the Intel I210 and I225. This version of DETD also provides flexibility for users to manually configure TAPRIO settings. While DETD in this extended state can be used for configuration of TSN Linux end-hosts, it is still not a production-grade solution. It is however a proof of concept of how a production-grade alternative could be developed.

5.2 Extensions to DETD

The extensions made to DETD for it to be a viable solution for TSN Linux end-host configuration are presented in this section. The validation of these extensions is also presented.

5.2.1 Add Listener

The add listener extension is one of the extensions made to DETD. In practice, this extension means that DETD can be used to configure a Linux end-host to listen for traffic sent with a specified multicast MAC-address.

The functionality for adding listeners was validated through a test consisting of two different stages. The intention of the test is to confirm that the listener configuration made by DETD correctly subscribes to the given multicast address. The experiment has two phases to confirm this. The first phase involves configuring the talker side through DETD while not applying any DETD configuration on the listener side. Txrx-tsn is set up on the talker side to transmit packets, and on the listener side to receive packets. The listener does not receive these packets. After this, the listener configuration from DETD is applied to the listener, and sending and receiving is repeated. This time the packets are received. This test confirms that the listener configuration is applied correctly by DETD, enabling the listener end-hosts to receive packets. The test also confirms that no mechanisms outside of DETD has enabled the end-host to receive multicast packets.

The second test verifies that the listener only has subscribed to the correct multicast MAC and does not receive traffic from any other multicast MAC addresses. This is done by having the listener and talker configurations from DETD applied to the boards and using txrx-tsn to transmit and receive packets in two phases. In the first phase packets are transmitted from the talker using a different multicast MAC than the one that the listener is subscribed to, which leads to packets not being received. In the second phase,

this is repeated with a multicast MAC address that matches the one that the listener side is configured to listen for. This confirms that the device is correctly configured to listen for the specified multicast MAC address and no other address.

5.2.2 Taprio Modes and Options

The TAPRIO modes and options extension provide a way to select one of the three different TAPRIO modes by selecting a flag when configuring talker streams in DETD. Additionally, the talker stream can be configured with customizable mapping.

To verify that the TAPRIO modes and mapping are applied according to the configuration provided to DETD as well as ensuring TAPRIO is applied correctly. Different configurations are executed one by one and verified by checking the terminal output of the Linux command tc qdisc show and comparing the output with the configuration given to DETD to determine if it was applied correctly. Comparing the output to the input parameters shows that the configuration is correctly applied.

5.2.3 I210 Support

The I210 support enables the option to configure the talker streams on the I210 NIC. The workflow of device configuration remains unchanged to ensure device abstraction.

To verify the support for the I210 NIC, different configurations which the device supports and does not support are executed and verified by comparing terminal output of the Linux command tc qdisc show with the configuration given and expecting errors on the configurations the device does not support. The supported configuration's output is compared to the input parameters showing the configuration is applied correctly. Attempting to apply unsupported configurations returns errors as expected, verifying that these configurations are not incorrectly applied.

Chapter 6

Conclusions

This chapter discusses the success of the features implemented in the project and the problems encountered. Possible future work within the subject is also discussed.

6.1 Discussion

The aim of this thesis work was to ease the task of configuring Linux end-hosts in IEEE 801.2 time-sensitive networks by providing a tool for Linux end-host configuration that is network interface card vendor independent and hides hardware complexity, where no such solution was previously available. This goal is reached by providing an extended version of the software DETD. These extensions are support for configuring listener streams, support for the Intel I210 network interface and additional configurability of the Linux TAPRIO queueing discipline.

In the most basic sense, the addition of listener stream support in DETD allows its use as a solution for TSN end-host configuration since it allows DETD to configure both listener and talker end-hosts. DETD in itself provides abstraction from hardware specific aspects of configuration, and its network device independence is extended through extending the list of supported network interfaces to include the Intel I210.

The extensions made to DETD were intended to be merged to the DETD GitHub repository. However, at this time merging has not yet been done.

6.1.1 Problems

One of the first problems encountered in the project was that the single-board computers designated for the project were not available at the planned time. The machines were delivered later than expected, and one machine lacked a cooling system causing further delays. Another problem occurred relatively early when installing DETD and caused additional time delays. This was a problem where DETD's installation would not work properly on certain Linux distributions. Reaching this conclusion and working around it required some troubleshooting. At the step where traffic generation was used to test the network, the use of a switch was originally intended. However, a problem appeared where packets from txrx-tsn were blocked. Since time was very limited at this point, it was decided to not use the switch in any of the experiments.

6.2 Conclusions

We have succeeded with the goal of easing the configuration of Linux TSN end-hosts with the features implemented to DETD, such as being able to configure listener streams, enabling customization of qdisc configuration, and providing support for the I210 NIC.

Even though more can be done to improve DETD, our solution is a good start to further develop from and shows the potential of DETD to configure Linux TSN end-hosts where future work could be done to add more features or provide additional ways to simplify the setup process. The extended version of DETD is also not a production-grade software, but rather a proof of concept to showcase how a TSN configuration tool could be implemented and for experimentation.

6.3 Future Work

Several improvements can be made to DETD. Some of these are listed in the DETD GitHub repository[5]. The testbed could also be extended to enable more complex experiments.

6.3.1 Integration of DETD in a larger TSN

It would be meaningful to implement DETD as a step in configuring a larger TSN in coordination with a CNC and possibly a CUC. This would involve a solution for automating the execution of DETD rather than running it independently on each machine. This would also open up the possibility to test DETD on more complex switched networks.

6.3.2 Supporting More Devices

Additionally, this thesis is limited to the use of Intel I225 network interface cards and partially of Intel I210 network interface cards. A feature that may be beneficial to add to DETD is the support for more network cards to further expand its practicality and decrease the hardware requirement limit. DETD already contains some class templates for not-yet implemented devices such as the Intel I226.

6.3.3 Integrating Time-Synchronization

A feature that will improve the usability of DETD is the integration of time synchronization. In the current state, DETD has no direct interaction with time synchronization. If it is included, DETD would have the capability to check if time synchronization is running. If it is not, DETD would have the possibility to start it and potentially tune the configuration.

6.3.4 Diagnostics Mode

On top of that, another useful feature is a mode for diagnostics. There are a few benefits to implementing a diagnostics mode in DETD. The first one is that troubleshooting issues with the device or misconfigurations of the device would be made easier through more information collected when running DETD. Additionally, issues relating to the time synchronization of the device can be provided to the user. With the diagnostics mode, timestamping could also be included. This would let the user do performance benchmarking on the applied configuration and provide an automated way to derive latency between two configured machines, which leads to the possibility to test different configurations and evaluate their performance in an effective way. All three of these benefits also show another benefit, which is that the information will be accessible from a single source.

Bibliography

[1] Detd documentation (figure). https://detd.readthedocs.io/en/latest/. Accessed: 2023-09-24.

[2] Lucia Lo Bello and Wilfried Steiner. A perspective on ieee time-sensitive networking for industrial communication and automation systems. *Proceedings of the IEEE*, 107(6):1094–1120, 2019.

[3] David Lou, Ulrich Graf, and Mitch Tseng. Virtualized programmable logic controllers. https://www.controleng.com/articles/virtualized-programmable-logic-controllers/, October 2021. Accessed: 2023-06-19.

[4] Opencnc_demo gitlab. https://git.cse.kau.se/hamzchah/opencnc_demo. Accessed: 2023-07-07.

[5] Detd github. https://github.com/Avnu/detd/blob/master/README.md. Accessed: 2023-05-06.

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

[7] 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)*, pages 1–57, 2016.

[8] Tsn wikipedia. `https://en.wikipedia.org/wiki/Time-Sensitive_Networking`. Accessed: 2023-04-28.

[9] Tc linux manual page. `https://www.man7.org/linux/man-pages/man8/tc-prio.8.html`. Accessed: 2023-04-06.

[10] Tc-taprio linux manual page. `https://man7.org/linux/man-pages/man8/tc-taprio.8.html`. Accessed: 2023-04-06.

[11] Tc-etf linux manual page. `https://man7.org/linux/man-pages/man8/tc-etf.8.html`. Accessed: 2023-04-10.

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

[13] Protocol buffers documentation. `https://protobuf.dev/`. Accessed: 2023-04-17.

[14] Ip-link linux manual page. `https://man7.org/linux/man-pages/man8/ip-link.8.html`. Accessed: 2023-04-17.

[15] Ethtool linux manual page. `https://www.man7.org/linux/man-pages/man8/ethtool.8.html`. Accessed: 2023-06-06.

[16] Ip linux manual page. `https://man7.org/linux/man-pages/man8/ip.8.html`. Accessed: 2023-06-06.

[17] Txrx-tsn github. https://github.com/intel/iotg_tsn_ref_sw/tree/0ee32cd0f38b8bb0451445c1f06bd7907b23ef1d. Accessed: 2023-05-06.

[18] Debian installer bookworm alpha 2 release. https://www.debian.org/devel/debian-installer/News/2023/20230219. Accessed: 2023-05-06.

[19] Kernel.org. https://mirrors.edge.kernel.org/pub/linux/kernel/projects/rt/6.1/. Accessed: 2023-05-06.

[20] Mac address wikipedia. https://en.wikipedia.org/wiki/MAC_address. Accessed: 2023-09-24.

Attachments

Appendix A

Abbreviations

ETF - Earliest txtime first

TAPRIO - Time aware priority shaper

TSN - Time-sensitive network

PCP - Priority code point

Qdisc - Queueing discipline