Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Erweiterung des Mininet Netzwerk-Emulators um einen zeitgesteuerten Scheduling-Mechanismus

Jona Herrmann

Studiengang: Informatik

Prüfer/in: Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel

Betreuer/in: Dr. rer. nat. Frank Dürr
M. Sc. David Hellmanns

Beginn am: 14. Oktober 2020

Beendet am: 14. April 2021

Kurzfassung

In der Industrie 4.0 spielt deterministische Echtzeitkommunikation eine immer größere Rolle. Bisher wurden in der Industrie Feldbussysteme eingesetzt. Diese werden heutzutage durch die Standard-Ethernet-Technologie ersetzt. Da aber Ethernet ursprünglich keine Echtzeitfähigkeit besaß, wurde Ethernet um den Time-sensitive Networking (TSN) Standard erweitert. Dieser definiert verschiedene Verkehrsklassen und ein Zeitmultiplexing (engl. Time-division Multiple Access, TDMA), wodurch deterministische Echtzeitkommunikation erreicht werden kann.

TSN-Netze können in Hardware-Testbeds oder mit Simulationen getestet werden. Der Nachteil dieser Möglichkeiten ist, dass die TSN-Hardware teuer ist und Simulationen ein abstrahiertes Modell erfordern. Deshalb wird nach einer besseren Möglichkeit zum Testen gesucht. Eine vielversprechende Methode ist dabei die Netzwerk-Emulation. Damit lassen sich reale Anwendungen durch in Software emulierter TSN-Hardware testen.

Ziel dieser Arbeit ist daher die Erweiterung des aus dem Software-defined Networking-(SDN)-Umfeld bekannten Mininet Netzwerk-Emulators um TSN-Funktionalität, insbesondere den Time-aware Shaper (TAS). Hierfür soll die Time-aware-Priority-Shaper (TAPRIO) Queueing-Discipline (Qdisc), welche eine Linux-Software-Implementierung des TAS ist, in Mininet integriert werden. Mithilfe von Mininet können somit Link-Eigenschaften wie Link-Verzögerung und Datenrate und mit TAPRIO das TSN-Scheduling emuliert werden, sodass auf einem Rechner TSN-Netze analysiert und getestet werden können. Es werden dazu verschiedene Design-Alternativen verglichen und implementiert sowie deren Eigenschaften in Experimenten analysiert.

Inhaltsverzeichnis

1 Einleitung **13**

2 Grundlagen **15**
2.1 Time-Sensitive Networking 15
2.2 Linux-Network-Stack 18
2.3 Traffic-Control 21
2.4 ip Command-Line-Programm 26
2.5 Mininet 28

3 Verwandte Arbeiten **33**

4 Problemstellung **35**

5 Design **37**
5.1 Erweiterung Veth um mehrere TX-Queues 37
5.2 Ermöglichen der Funktion von TAPRIO mit Veth 37
5.3 TAPRIO und Mininet-Qdisc 38

6 Implementierung **51**
6.1 Erweiterung Veth um mehrere TX-Queues 51
6.2 Erweiterung Mininet-CLI 51
6.3 TAPRIO und Mininet-Qdisc 53

7 Evaluation **57**
7.1 Durchführung der Messungen 57
7.2 Leistungsfähigkeit Veth 57
7.3 Mininet ohne Mininet-Qdisc 58
7.4 TAPRIO_MININET Qdisc 59
7.5 Mininet mit Scheduling und Link-Verzögerung 71

8 Zusammenfassung und Ausblick **75**

Literaturverzeichnis **77**

Abbildungsverzeichnis

2.1 IEEE 802.1Q Header 16
2.2 TAS 17
2.3 Aufbau Linux-Network-Stack für das Senden 19
2.4 Beispiel einer verschachtelten Qdisc 24
2.5 Interner Aufbau von TAPRIO an Interface mit vier TX-Queues 25
2.6 Network-Namespaces mit veth und Software-Switch 28
2.7 Beispiel Netzwerkaufbau mit Mininet 29
2.8 Beispiel Kombination der Mininet-Qdisc 30

4.1 Mininet Aufbau mit TAPRIO 35

5.1 Reale Situation im TSN-Netzwerk und entsprechende Umsetzung mit Mininet . . 39
5.2 Option TAPRIO unter Mininet-Qdisc 40
5.3 Option TAPRIO vor Mininet-Qdisc 42
5.4 Aufbau der TAPRIO_EXTEND Qdisc 44
5.5 Aufbau der TAPRIO_MININET Qdisc 45
5.6 Funktionsweise der TAPRIO_MININET Qdisc 46
5.7 Funktionsweise TAPRIO_MININET mit Root-Qdisc zur Datenratenbegrenzung . 48
5.8 Aufbau TAPRIO_MININET mit Kind-Qdiscs von TAPRIO zur Datenratenbegrenzung 49

7.1 Setup Veth 58
7.2 Übertragungszeit mit Veth 58
7.3 Histogramm der Ausreißer bei Veth 59
7.4 Mininet Setup 60
7.5 Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket mit Mininet Setup 61
7.6 Grundlegendes Setup für TAPRIO_MININET 61
7.7 Vergleich der Verzögerungszeit von TAPRIO mit TAPRIO_MININET 62
7.8 Histogramm der Ausreißer beim Vergleich der Verzögerungszeit 62
7.9 Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket mit TAPRIO_MININET Setup 63
7.10 Abweichung von der eingestellten Link-Verzögerung mit Netem in Kombination mit TAPRIO_MININET 64
7.11 Histogramme der Ausreißer von TAPRIO_MININET mit Netem 65
7.12 Begrenzung der Datenrate in Kombination mit TAPRIO_MININET auf 100 MBit/s 66
7.13 Ergebnisse TAPRIO_MININET mit Scheduling und Verzögerung 67
7.14 Ergebnisse Qdisc zur Datenratenbegrenzung in Mininet-Qdisc 68
7.15 Ergebnisse Datenratenbegrenzung vor TAPRIO_MININET 69
7.16 Ergebnisse Datenratenbegrenzung durch TAPRIO Kind-Qdiscs jeweils auf 100 MBit/s eingestellt 70

7.17 Ergebnisse Datenratenbegrenzung durch TAPRIO Kind-Qdiscs jeweils auf 50 MBit/s eingestellt 71
7.18 Ergebnisse Datenratenbegrenzung durch TAPRIO_MININET 72
7.19 Datenratenbegrenzung durch TAPRIO_MININET mit Intervallen von 1 ms . . . 72
7.20 Ergebnisse Datenratenbegrenzung durch TAPRIO_MININET 73
7.21 Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket 74

Verzeichnis der Listings

2.1 Methodenaufrufe der Qdisc im Linux-Network-Stack 20
2.2 Beispiel tc Add TAPRIO Befehl 26
2.3 Beispiel ip Add Veth Befehl 27
5.1 Add veth ip-Befehl mit Parameter numtxqueues 37
5.2 Hinzugefügte Filter-Befehle für das Mininet-CLI 38
5.3 Hinzugefügte TAPRIO-Befehle für das Mininet-CLI 43
6.1 Anpassung TAPRIO Qdisc 53

Abkürzungsverzeichnis

CBS Credit-Based-Shaper. 17

CLI Command-Line-Interface. 31

DES Discrete Event Simulation. 13

FIFO First In – First Out. 18

GCL Gate Control List. 16

HFSC Hierarchical-Fair-Service-Curve. 23

HTB Hierarchy-Token-Bucket. 23

IQA Interquartilsabstand. 57

KA Kritischer Abschnitt. 40

LAN Local Area Network. 15

lo Loopback. 37

netem Network-Emulator. 22

netns Network-Namespace. 26

OVS Open vSwitch. 29

PCP Priority-Code-Point. 16

Qdisc Queueing-Discipline. 13

red Random-Early-Detection. 22

RX Receive. 20

SDN Software-defined Networking. 14

SKB Socket-Buffer. 18

TAPRIO Time-aware-Priority-Shaper. 13

TAS Time-aware Shaper. 13

tbf Token-Bucket-Filter. 22

tc Traffic-Control. 21

TC Traffic Class. 16

TDMA Time-division Multiple Access. 13

TSN Time-sensitive Networking. 13

TX Transmit. 18

UDP User Datagram Protocol. 38

veth Virtual Ethernet Interface. 26

VLAN Virtual Local Area Network. 15

1 Einleitung

In der Industrie 4.0 spielt Echtzeitkommunikation eine immer wichtigere Rolle, insbesondere für sicherheitskritische Anwendungen wie Steuerung von Robotern und Maschinen. Dabei spielt die Sicherheit dieser Systeme eine große Rolle. Insbesondere im möglichen Fehlerfall. Zum Beispiel ist es bei der automatisierten Produktion mit Roboterarmen wichtig, falls ein Fehler auftritt, beispielsweise der Roboterarm falsche positioniert ist, dass auf diesen in Echtzeit reagiert werden kann. Geschieht das nicht, kann das große wirtschaftliche Folgen haben. Roboterarme können beschädigt oder zerstört werden und damit die gesamte Produktion aufgehalten werden. Aber es können auch die gerade produzierten Bauteile zerstört werden. Um das zu vermeiden, ist Echtzeitkommunikation zwischen Sensoren und der dazugehörigen Steuerungseinheit nötig. Dabei muss die Steurungseinheit rechtzeitig die tatsächliche Position des Roboterarmes von den Sensoren bekommen. Somit kann in diesem Beispiel die Position des Roboterarmes schnell korrigiert werden, damit es zu keiner Beschädigung kommen kann. Dementsprechend müssen auch die Anweisungen der Steuerungseinheit den Roboterarm in Echtzeit erreichen.

Bisher wurden in der Industrie spezialisierte Netze, die sogenannten Feldbusse eingesetzt. Diese sollen heutzutage durch die Standard-Ethernet-Technologie ersetzt werden. Da aber Ethernet ursprünglich nicht echtzeitfähig war, wurde Ethernet durch den Time-sensitive Networking (TSN) Standard erweitert. Mit TSN wird im Ethernet-Netzwerk deterministische Echtzeitkommunikation ermöglicht. Damit ist es möglich, Echtzeitkommunikation als wichtigen Baustein zu verwenden. Eine wichtige Komponente von TSN ist der sogenannte Time-aware Shaper (TAS), welcher grundlegend ein Time-division Multiple Access (TDMA) Schema implementiert. Dieser teilt den Paketverkehr anhand der in den Paketen definierten Priorität in acht verschiedene Queues ein. Hinter jeder Queue befindet sich ein Gate. Zu welchem Zeitpunkt ein bestimmtes Gate offen ist, wird durch einen definierten Zyklus gesteuert. Wenn ein Gate offen ist, können Pakete aus dieser Queue gesendet werden. Somit wird ein zeitgesteuertes Scheduling mit harten Echtzeitgarantien realisiert. Eine Software-Implementierung eines TASs für Linux ist die Time-aware-Priority-Shaper (TAPRIO) Queueing-Discipline (Qdisc).

Doch bevor solche Echtzeit-Netzwerke in teurer Hardware realisiert werden, wäre es wichtig, diese testen beziehungsweise evaluieren zu können, ob die Echtzeitgarantien überhaupt gewährleistet werden können. Eine Möglichkeit dazu wären Hardware-Testbeds. Doch da die TSN-Hardware teuer ist, können nur kleine Testbeds realisiert werden. Somit ist diese Möglichkeit nicht skalierbar und wegen der hohen Kosten auch oft nicht verfügbar.

Eine andere Möglichkeit ist die Simulation. Bei dieser steht das Ergebnis im Vordergrund. Das bedeutet, dass das Problem mit einem abstrahierten Modell simuliert wird. Durch diese Abstraktion entspricht die Funktionsweise des Simulators selten der Funktionsweise des realen Problems. Somit ist die Simulation keine reale Anwendung. Außerdem können sehr lange Laufzeiten entstehen und diese werden noch größer, wenn die Genauigkeit erhöht werden soll. Ein möglicher Ansatz für die Simulation von TSN-Netzwerken ist mit den Discrete Event Simulation (DES) Modellen.

Hierbei wird das Verhalten der Netzwerkkomponenten nur an wichtigen Zeitpunkten simuliert. Die DES-Modelle können helfen, solche Netzwerke besser zu verstehen. Doch da diese Modelle einfach gehalten sind, wird nach besseren Möglichkeiten zum Testen gesucht.

Eine weitere Möglichkeit wäre, das TSN-Netzwerk zu emulieren. Bei der Emulation kommt es dabei im Gegensatz zur Simulation auch auf die Funktionsweise an. Diese sollte dem realen Problem sehr ähnlich sein, am besten genau so funktionieren. Ein bekannter Emulator für Netzwerke ist der Mininet Netzwerk-Emulator. Dieser wurde ursprünglich für die Emulation von Software-defined Networking (SDN)-Netzen entwickelt. Dabei können beliebig große Netze mit definierten Datenraten und Link-Verzögerungen emuliert werden. Die Vorteile von Mininet sind, dass die Implementierung quelloffen und mit gängigen Betriebssystemen kompatible ist. Doch Mininet bietet bisher keine Echtzeitunterstützung, insbesondere kein TSN. Gagan Nandha Kumar at al. [8] haben Messungen mithilfe von TAPRIO in Mininet durchgeführt. Da aber in diesem Paper nicht genau auf die Realisierung eingegangen wird, ist nicht klar, ob diese Lösung allgemeingültig ist oder nur für den speziellen Testfall funktioniert.

Deshalb ist das Ziel dieser Bachelorarbeit, die Erweiterung von Mininet um einen zeitgesteuerten Scheduling-Mechanismus. Dabei soll die vollständige Funktionalität von Mininet, also mit allen Konfigurationen, erhalten bleiben. Realisiert werden soll der zeitgesteuerte Scheduling-Mechanismus, indem die TAPRIO Qdisc in Mininet integriert wird. Mit der entstanden Erweiterung können TSN-Netzwerke relativ einfach und kostengünstig analysiert werden. Ein großer Vorteil dabei ist die Flexibilität. Dadurch können die erzeugten Netzwerke sehr leicht angepasst werden. Außerdem sind durch die realitätsnahe Funktionsweise des Emulators die Ergebnisse realistisch und sehr gut auf das reale Netzwerk übertragbar.

Im Folgenden Kapitel 2 wird ein Überblick über die verwendeten Technologien in dieser Arbeit gegeben, welche für das Verständnis wichtig sind. Danach wird in Kapitel 3 auf verwandte Arbeiten in diesem Themenfeld eingegangen. In Kapitel 4 werden die Probleme aufgezählt, welche gelöst werden müssen, damit TAPRIO in Kombination mit der Emulation der Link-Eigenschaften in Mininet genutzt werden kann. Für diese werden dann in Kapitel 5 Design-Alternativen entwickelt. Letztendlich wird in Kapitel 6 die Implementierung dieser beschrieben. Anschließend wird in Kapitel 7 die implementierten Lösungen evaluiert, ob die geforderten Eigenschaften damit realisiert werden können. In Kapitel 8 werden die Ergebnisse zusammengefasst und ein Ausblick auf anschließende Arbeiten gegeben.

2 Grundlagen

In diesem Kapitel werden die grundlegenden Technologien eingeführt, welche für das Verständnis dieser Arbeit notwendig sind.

2.1 Time-Sensitive Networking

Ethernet ist eine verbreitete Technologie für die Netzwerkkommunikation in Local Area Networks (LANs). Das Institute of Electrical and Electronics Engineers (IEEE) hat diese im Standard IEEE 802.3 definiert. Ein Ethernet-Netzwerk besteht dabei hauptsächlich aus Hosts und Switches. Mit den Switches kann ein großes LAN mit vielen Hosts aufgebaut werden. Dabei wird zwischen zwei grundlegende Arten von Switches unterschieden: Hardware-Switches und Software-Switches. Der Unterschied zwischen diesen, dass der Hardware-Switch einen physischen Switch darstellt, dieser also in Hardware realisiert ist. Der Software-Switch im Gegensatz dazu nur eine Software ist, welche auf einem Rechner läuft. Beide Arten realisieren aber dennoch die gleiche Funktionalität. Diese ist das Weiterleiten von Frames. Da ein Switch mehrere Ports besitzt, an denen entweder Hosts oder weitere Switches angeschlossen sind, muss dieser wissen, wie das Frame weitergeleitet werden soll. Wenn ein Frame über ein Port, welcher der Ingress Port ist, in ein Switch kommt, leitet dieser es über einen anderen Port, den Egress Port, weiter. Dabei wird der Port zum Beispiel so gewählt, dass das Frame über den kürzesten Pfad durch das Netzwerk zum Ziel gelangt. Dies kann gelöst werden, indem der Switch jeweils die Quell-MAC-Adresse lernt. Dazu gibt es eine Tabelle, in welcher zu jeder Ziel-MAC-Adresse der jeweilige Port gespeichert ist, über welchen das Frame gesendet werden soll.

Da Ethernet ursprünglich nicht echtzeitfähig war, wurde eine Sammlung von IEEE-Standards definiert und unter dem Begriff TSN zusammengefasst. TSN ermöglicht deterministische Echtzeitkommunikation in Ethernet-Netzwerken. Dafür sind zwei grundlegende Mechanismen erforderlich. Es muss die Möglichkeit geben, jedem Frame eine Priorität zu zuweisen, um damit priorisierten Verkehr zu ermöglichen. Außerdem muss das Scheduling definiert werden, also die Entscheidung, welches Frame als Nächstes gesendet wird.

2.1.1 Priorisierung

Um priorisierten Netzwerkverkehr zu ermöglichen, muss es die Möglichkeit geben, jedem Paket eine Priorität zuzuweisen. Dafür wird der optionale Virtual Local Area Network (VLAN)-Tag verwendet, welcher im Standard IEEE 802.1Q standardisiert ist. Wie in Abbildung 2.1 zusehen ist, wird dabei der Ethernet-Frame-Header nach der Quell-MAC-Adresse um ein 32 Bit Feld, dem VLAN-Tag, erweitert. Dieser setzt sich aus dem Tag-Protocol-Identifier (TPID) und der Tag-Control-Information (TCI), welche jeweils 16 Bit groß sind, zusammen. Der TPID hat immer den festen Wert 0x8100, der

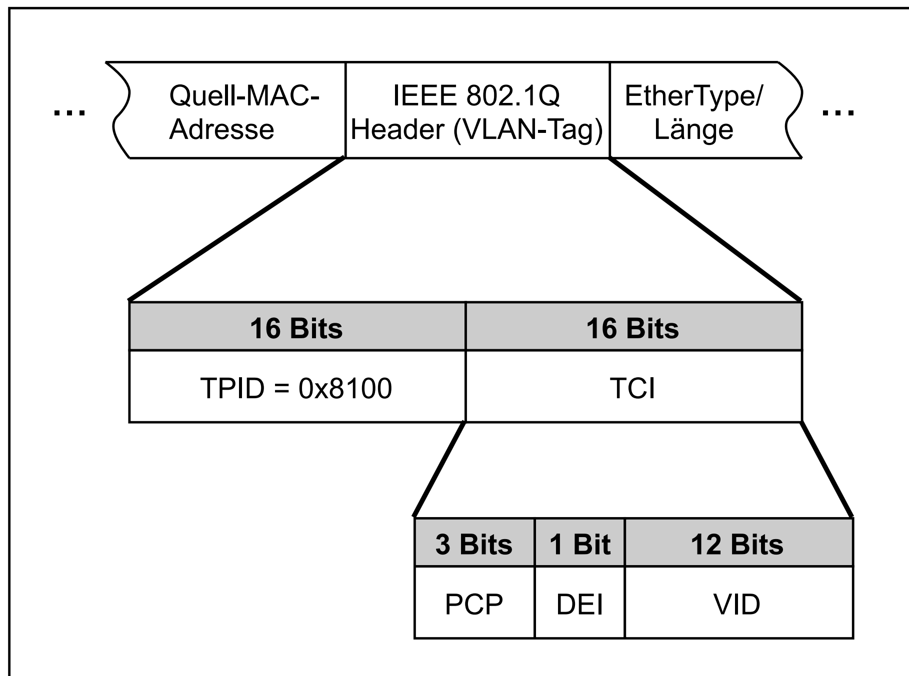


Abbildung 2.1: IEEE 802.1Q Header

anzeigt, dass das Frame einen VLAN-Tag besitzt. Die TCI setzt sich wiederum aus dem 3 Bit großen Priority-Code-Point (PCP)-Feld, dem 1 Bit großen Drop-Eligible-Indicator (DEI) Feld und dem 12 Bit großen VLAN-Identifier (VID) zusammen. Für TSN ist nur das PCP-Feld von Bedeutung, da mit diesem der Benutzer jedem Frame eine Priorität zwischen null und sieben zuweisen kann.

2.1.2 Scheduling

Beim Scheduling wird entschieden, wann Frames unterschiedlicher Priorität gesendet werden. Das einfachste Scheduling ist durch Strict-Priority Queueing. Dabei werden immer die Pakete mit der höchste Priorität zuerst gesendet. Somit kann dadurch keine harten Echtzeitgarantien gewährleistet werden. Damit das Scheduling für TSN realisiert werden kann, wird als Grundlage ein TDMA benötigt. Dadurch kann jeder Priorität einen gewissen Zeitslot zugeteilt werden, in diesem nur Frames dieser Priorität gesendet werden. Eine mögliche Implementierung dafür ist der TAS, welcher in IEEE 802.1Qbv standardisiert ist.

Ein TAS wird an allen Ports eins Switches gesetzt. In Abbildung 2.2 ist der Aufbau an einem Port mit gesetztem TAS dargestellt. Die eintreffenden Pakete werden bei der Klassifizierung anhand ihres PCP-Wertes in acht verschiedene Traffic Classes (TCs) eingeteilt. Für jede TC gibt es eine Queue, in dieser befinden sich alle Frames der jeweiligen TC. Hinter jeder Queue befindet sich ein Gate. Wann ein Gate offen oder geschlossen ist, wird in der sogenannten Gate Control List (GCL) definiert. Die GCL besteht aus Einträgen, welche den Zustand aller Gates definiert. Die Gatezustände werden dabei mit einem Bit-Vektor dargestellt, wobei jede Bit-Position ein Gate beschreibt. Dabei bedeutet eine 1, dass das Gate offen ist und eine 0, dass das Gate geschlossen ist. Außerdem wird in jedem GCL Eintrag noch eine Zeit definiert. Diese gibt an, wann zum nächsten Eintrag gegangen wird. Wenn

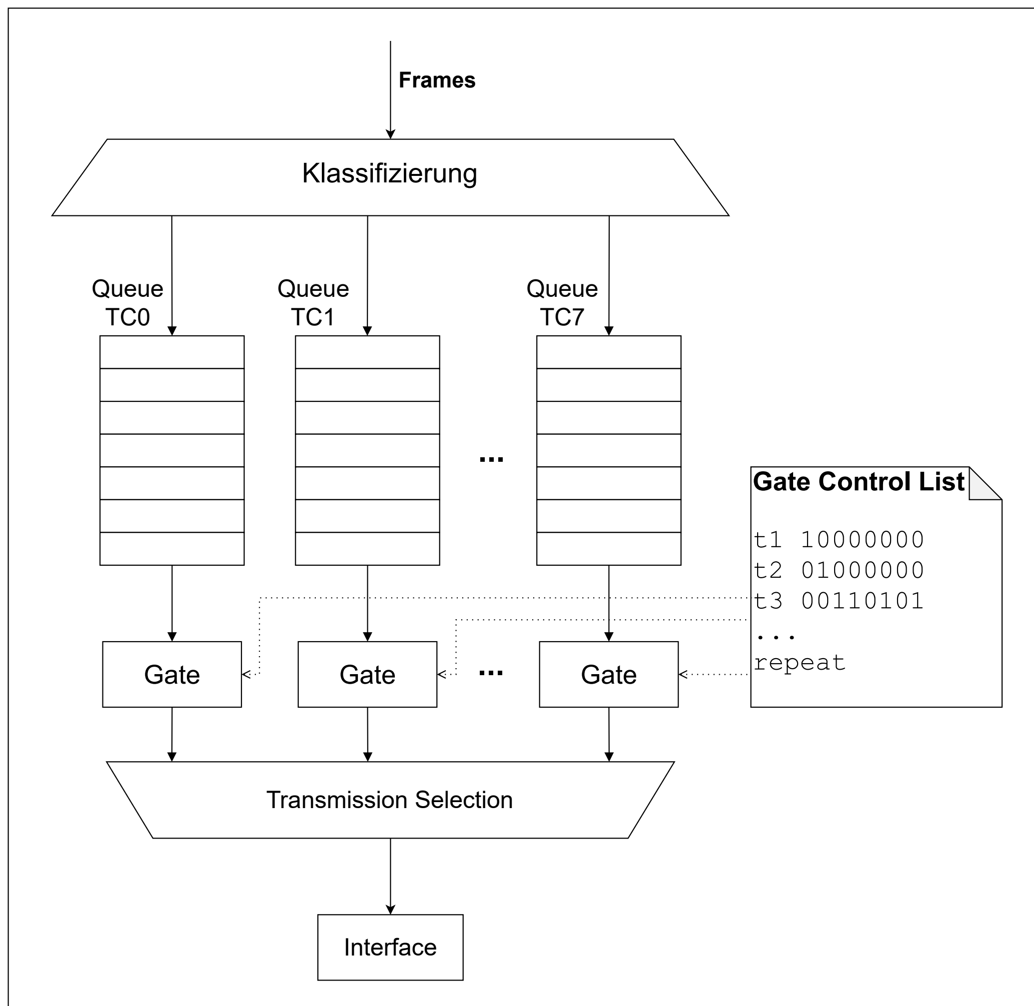


Abbildung 2.2: TAS

der letzte Eintrag erreicht wird, wird wieder mit dem ersten begonnen. Somit wird die GCL zyklisch durchlaufen. Wenn ein Gate offen ist, kann aus der jeweiligen Queue ein Paket entnommen und über das Interface gesendet werden. Da der Standard aber erlaubt, dass mehrere Gates zur gleichen Zeit geöffnet sind, also in einem GCL Eintrag mehrere Bits auf 1 gesetzt sind, muss entschieden werden, welches Frame dann gesendet wird. Dafür gibt es die Transmission Selection, welcher ein weiterer Scheduling-Algorithmus ist. Für diese können verschiedene Algorithmen verwendet werden ein Beispiel wäre der Strict-Priority Queueing Ansatz. Dieser wählt dann letztendliche das Frame aus, welches über das Interface gesendet wird. Es gibt noch weitere IEEE-Shaper welche noch zusätzlich genutzt werden können zum Beispiel den Credit-Based-Shaper (CBS), welcher in IEEE 802.1Qav standardisiert ist. [1]

2.2 Linux-Network-Stack

Jedes Betriebssystem, welches Kommunikation über ein Netzwerk nutzt, benötigt einen Network-Stack. Dieser besteht aus verschiedenen Schichten, welche beschreiben wie die zu sendenden Pakete von der Applikation bis zum Interface verarbeitet werden. Aber auch wie empfangene Pakete auf einem Interface verarbeitet werden und schließlich die Applikation erreichen. Da in dieser Arbeit Linux als Betriebssystem verwendet wird, wird in diesem Abschnitt der Linux-Network-Stack genauer beschrieben.

In Abbildung 2.3 ist der Linux-Network-Stack für das Senden eines Paketes dargestellt. Der Benutzer kann in seiner Applikation, welche im User-Space läuft, mit den Systemcalls *sendto()*, *sendmsg()* oder *write()* Pakete senden. Diese werden zunächst in einem Sende-Socket-Buffer gepuffert. Von dort aus gehen die Pakete durch den Transport-Layer- und Network-Layer-Stack. Dabei wird zuerst ein Socket-Buffer (SKB) angelegt. Dies ist eine Datenstruktur des Linux-Kernels, welche ein Paket repräsentiert. In dieser befinden sich Metadaten, wie eine 32 Bit Prioritätsfeld und Pointer auf die verschiedenen Paket-Header. Als Nächstes wird das Paket in die Egress Qdisc des Interfaces, über die das Paket gesendet werden soll, einsortiert. Diese definiert, in welcher Reihenfolge und zu welchem Zeitpunkt die Pakete letztendlich gesendet werden. Die einfachste Möglichkeit ist eine First In – First Out (FIFO)-Qdisc. In Abschnitt 2.3 wird genau erklärt, wie und welche Varianten hier genau konfiguriert werden können. Da die Qdisc in dieser Arbeit eine wichtige Rolle spielt, wird diese im nächsten Absatz noch genauer erklärt. Die entnommenen Pakete aus der Qdisc werden dann in die jeweilige Transmit (TX)-Queue des Interfaces eingefügt. Diese sind meist als Ring-Puffer implementiert. Von da werden die Pakete über das Interface gesendet.

Wie die Qdisc im Linux-Network-Stack funktioniert, wird anhand des Linux-Kernels 5.6.5 gezeigt, da diese Kernelversion in dieser Arbeit verwendet wird. Im Network-Layer-Stack wird mit der Methode *dev_queue_xmit()* die Qdisc aufgerufen. Diese befindet sich wie alle Methoden, welche im Name ein „dev“ haben in */net/core/dev.c*. In Listing 2.1 ist beginnend mit diesem Methodenaufruf eine Art Methodenstack für die Qdisc im Linux-Network-Stack dargestellt. Dabei ist nur eine grober Überblick über die genaue Funktionsweise dargestellt. Das bedeutet insbesondere, dass die Methoden nicht im Detail erklärt werden. Dabei liegt der Fokus auf der Standardfunktionsweise. Es wird also nicht auf alle möglichen Spezialfälle eingegangen. Die im Network-Layer-Stack aufgerufene Methode stellt dabei nur ein Wrapper für die Methode *__dev_queue_xmit()* dar. In dieser wird als Erstes eine Filterung vorgenommen, sofern diese definiert ist. Wie diese definiert werden kann siehe Abschnitt 2.3.1. Wenn die gesetzte Qdisc TX-Queues hat, wird die Methode *__dev_xmit_skb()* aufgerufen. In dieser Methode wird zwischen vier verschiedene Fällen unterschieden. Der erste Fall ist, wenn die Qdisc kein Locking benötigt. Auf diesen wird hier nicht weiter eingegangen. Alle anderen Fälle benötigen ein Locking, das mit der Methode *spin_lock()* erreicht wird. Als Nächstes wird geprüft ob die Qdisc überhaupt aktiviert ist, wenn nicht, wird das Paket gedroppt. Falls die Qdisc Bypassing, also das umgehen der Qdisc, erlaubt, leer ist und nicht läuft, wird das Paket sofort in die jeweilige TX-Queue des Interfaces eingefügt. Ansonsten wird das Paket in die gesetzte Qdisc mit *enqueue()* eingefügt. In Abschnitt 2.3.2 wird darauf eingegangen, wie dies genau funktioniert.

Danach wird mit *qdisc_run_begin()* geprüft, ob die Qdisc läuft. Diese Methode befindet sich in */include/net/sch_generic.h*. Falls ja, wird der if-Block übersprungen und der Systemcall zum Senden ist, nachdem noch der Lock freigegeben wurde, beendet. Wenn nicht, wird der Zustand der Qdisc auf läuft gesetzt und der if-Block wird ausgeführt. In diesem wird die Methode *__qdisc_run()* aufgerufen, welche ein While-Schleife ausführt. Dabei wird als Bedingung *qdisc_restart()* verwendet. Diese

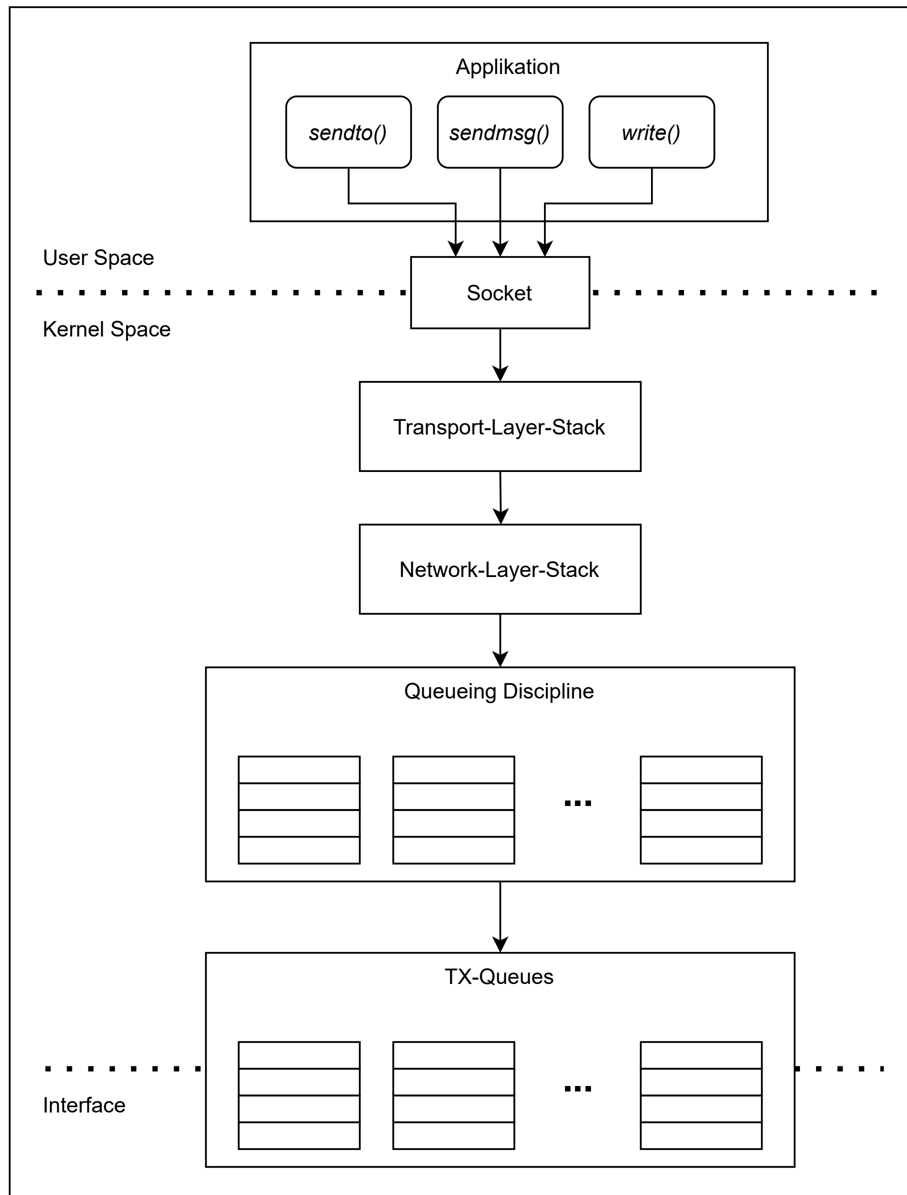


Abbildung 2.3: Aufbau Linux-Network-Stack für das Senden

ist wie *__qdisc_run()* in */net/sched/sch_generic.c* implementiert. In *qdisc_restart()* wird zuerst versucht, ein Paket mit *dequeue_skb()* aus der gesetzten Qdisc zu entnehmen. In Abschnitt 2.3.2 wird dies genauer erläutert. Wenn kein Paket entnommen werden kann, wird *False* zurückgegeben und die Schleife in *__qdisc_run()* wird abgebrochen. Ansonsten wird die TX-Queue ausgewählt, in welche das Paket eingefügt werden soll. Danach wird *sch_direct_xmit()*, welche auch in */net/sched/sch_generic.c* definiert ist, ausgeführt. Diese Methode fügt letztendlich das Paket der ausgewählten TX-Queue hinzu. Von dort aus wird es dann über das Interface gesendet. Der Rückgabewert von *sch_direct_xmit()* wird direkt auch von *qdisc_restart()* zurückgegeben. Dabei ist dieser *True* wenn das Senden erfolgreich war und *False* wenn nicht. Wenn das Senden fehlgeschlagen ist, wird das Paket wieder der gesetzten Qdisc hinzugefügt.

Listing 2.1 Methodenaufrufe der Qdisc im Linux-Network-Stack

```
dev_queue_xmit()
    __dev_queue_xmit()
    |   tc Filterung
    |   __dev_xmit_skb()
    |   |   /* Fall Qdisc keine Work-conserving Qdisc und braucht Locking
    |   |   */ (Flag TCQ_F_CAN_BYPASS und TCQ_F_NOLOCK nicht gesetzt)
    |   |   spin_lock(Qdisc)
    |   |   Qdisc enqueue()
    |   |   if (qdisc_run_begin()) {
    |   |       __qdisc_run(q)
    |   |       |    while (qdisc_restart()) {
    |   |       |       |   dequeue_skb()
    |   |       |       |   TX-Queue wird ausgewaehlt
    |   |       |       |   sch_direct_xmit()
    |   |       |       if (quota <= 0) {
    |   |       |           __netif_schedule()
    |   |       |           break
    |   |       |       }
    |   |       |    }
    |   |       qdisc_run_end()
    |   |   }
    |   |   spin_unlock(Qdisc)
```

Jetzt kommen wir noch mal zu der Schleife in *__qdisc_run()*. Denn solange Pakete erfolgreich gesendet werden, wird dies Schleife nie abbrechen und somit den Lock nie freigeben. Dies führt dazu, dass in dieser Zeit keine neuen Pakete mehr in die Qdisc mit *enqueue()* eingefügt werden können, da dazu der Lock benötigt wird. Um das zu vermeiden, gibt es die Variable *quota*, welche mit einem Interface spezifisches Gewicht außerhalb der Schleife initialisiert wird. Diese wird nach jedem erfolgreichen Senden um die Anzahl der gesendet Pakete dekrementiert. Wenn diese Variable kleiner gleich null ist wird *__netif_schedule* aufgerufen. Diese löst einen *NET_TX_SOFTIRQ* Interrupt aus und danach wird die Schleife abgebrochen. Dieser Interrupt ruft letztendlich wieder *__qdisc_run()* auf. Somit können dann wieder Pakete gesendet werden. Nachdem die Schleife abgebrochen ist, wird in die Methode *__dev_xmit_skb()* zurückgekehrt. Dort wird dann *qdisc_run_end()* ausgeführt. Diese setzt den Zustand der Qdisc auf, läuft nicht. Danach wird der Lock noch freigegeben und alle Methodenaufrufe bis hin zum Systemcall fürs Senden werden beendet.

Vollständigkeit halber wird auch noch der Linux-Network-Stack für das Empfangen eines Pakets kurz erläuterte. Wenn ein Paket auf einem Interface empfangen wird, wird dieses in einer Receive (RX)-Queue gepuffert. Diese sind meist wieder als Ring-Puffer implementiert. Dabei wird für jedes empfangen Paket ein SKB erstellt. Danach geht das Paket durch die Ingress Qdisc, falls diese auf dem Interface gesetzt wurde. Hier kann eine Filterung der Pakete vorgenommen werden, siehe dazu wieder Abschnitt 2.3.1. Als Nächstes durchläuft das Paket den Network-Layer- und Transport-Layer-Stack. Danach wird das Paket im Receive-Socket-Buffer zwischengespeichert. Mit den Systemcalls *recvfrom()*, *read()* oder *recvmsg()* kann das Paket von der Applikation aus dem Buffer entnommen werden.

2.3 Traffic-Control

Mit Traffic-Control (tc) [15] kann der Netzwerkverkehr im Linux-Kernel beeinflusst werden. Dazu gehört zum Beispiel Scheduling, welches das als nächste zu sendende Pakete auswählt. Aber auch Traffic-Shaping, womit der Verkehr kontrolliert werden kann. Dazu zählt zum Beispiel das verzögerte Weiterleiten von Paketen, dadurch kann die Datenrate begrenzt werden. Außerdem gibt es die Möglichkeit Pakete zu verwerfen (engl. drop).

Für die entsprechenden Beeinflussungen des Verkehrs gibt es drei grundlegende Komponenten: Qdisc, filter und class. Diese werden immer an einem Interface gesetzt und sind entweder Ingress, also beeinflussen Pakete, die den Kernel betreten oder Egress und beeinflusse Pakete, die den Kernel verlassen. Nicht alle Beeinflussungen können an Ingress und Egress gemacht werden zum Beispiel macht Scheduling an Ingress kein Sinn. Somit können Scheduling-Elemente nur an Egress gesetzt werden.

Für die Konfiguration von den drei oben erwähnten Komponenten befindet sich im Package iproute2 ein tc-Command-Line-Programm. In diesem Package sind weitere Programmen vorhanden, mit denen verschiedene Netwerkkonfigurationen im Kernel vorgenommen werden können. In dieser Arbeit wird Version 5.11.0 von iproute2 verwendet. Diese kann aus dem offiziellen Git-Repository [7] heruntergeladen werden. Alle Programme in diesem Package laufen im User-Space. Mit diesen kann über das Terminal die jeweiligen Konfigurationen vorgenommen werden. Da diese Programme im User-Space laufen, die Konfigurationen aber im Kernel erfolgen muss, wird Inter-process Communication (IPC) zwischen User-Space und Kernel-Space benötigt. Dafür wird Netlink verwendet, welches ein Linux-Protokoll für IPC ist. Der eingegebene Befehl in das Terminal wird zunächst von dem tc-Programm eingelesen. Daraus wird als Erstes eine tc-Nachricht generiert. Diese wird dann in eine Netlink-Nachricht eingebettet, welche über ein Socket in den Kernel gesendet wird, um dort die Konfiguration vorzunehmen.

2.3.1 Filter

Mit Filtern kann der Netzwerkverkehr gefiltert werden. Diese bestehen aus einem Match und mindestens einer Aktion. Der Match kann aus Teilen des gesamten Paket-Headers bestehen, zum Beispiel aus der Ziel-IP-Adresse. Eine Aktion gibt an, was gemacht werden soll, wenn ein Paket den Match erfüllt. Mögliche Aktionen sind die Modifikation des Paket-Headers, also zum Beispiel kann die Quell-IP-Adresse des Paketes geändert werden. Außerdem ist eine mögliche Aktion das droppen von bestimmten Paketen.

Mithilfe von Filtern können die Pakete an Ingress gefiltert werden. Dazu wird die Ingress Qdisc benötigt, welche eine spezielle Qdisc nur für das Hinzufügen von Filtern ist. Da man aber möglicherweise auch an Egress filtern will und dies mit der Ingress Qdisc nicht möglich ist, ist seit Kernel-Version 4.5 die clsact Qdisc vorhanden. Diese ist eine Weiterentwicklung der Ingress Qdisc, um zusätzlich auch noch an Egress filtern zu können. Somit ist die clsact Qdisc ein Spezialfall, weil diese an Ingress und Egress arbeitet. Damit bei den entsprechen gesetzten Filter definiert ist, wo diese filtern sollen, muss dies beim Setzten der Filter definiert werden. Das geht, indem beim Erzeugen noch zusätzlich Ingress oder Egress angegeben wird. Da mit der clsact Qdisc auch an Ingress gefiltert werden kann, wird empfohlen die Ingress Qdisc nicht mehr zu verwenden.

2.3.2 Queueing Discipline

Die Qdiscs sind für das Scheduling zuständig, also für das Auswählen des nächsten zu sendenden Paketes. Wie schon oben erwähnt macht Scheduling nur an Egress Sinn. Deshalb kann eine Qdisc auch nur an Egress gesetzt werden. Einzige Ausnahmen sind die oben erwähnten ingress und clsact Qdiscs, da diese kein Scheduling machen. Dabei wird die Egress Qdisc meist als Root-Qdisc bezeichnet. Es gibt viel verschiedene Qdiscs und jede hat eine bestimmte Funktionalitäten. Unterteilt werden können diese in Classless und Classful Qdiscs.

Classless Qdics

Classless Qdiscs sind, wie der Name schon sagt, klassenlos, also können an diesen keine Klassen definiert werden. Dementsprechend können an diesen Qdiscs auch keine Filter gesetzt werden. Es ist möglich, eine Sequenz von mehreren Classless Qdiscs zu definieren. Da aber keine Klassen existieren, durchläuft jeweils jedes Pakete alle gesetzten Qdiscs. Wie das Hintereinanderschalten realisiert wird und das enqueue und dequeue funktioniert, dazu siehe Abschnitt 2.3.2. Die simpelste Classless Qdisc ist eine FIFO-Qdisc. In Linux gibt zwei verschiedene Implementierungen einer FIFO-Qdisc: die pfifo und bfifo Qdisc. Diese beiden unterschieden sich nur dadurch, in welcher „Einheit" die Buffergröße angegeben wird. Dabei steht das „p" für Paket und das „b" für Byte. Weitere Beispiele für Classless Qdiscs sind die Token-Bucket-Filter (tbf) Qdisc [14], die Network-Emulator (netem) Qdisc [10] oder die Random-Early-Detection (red) Qdisc [12].

Classful Qdics

Der große Nachteil bei Classless Qdiscs ist, dass verschiedene Pakete nicht unterschiedlich behandelt werden können. Deshalb gibt es die Classful Qdiscs, denn diese können, wie der Name schon sagt, verschiedene Klassen unterscheiden. Somit kann jede Klasse anders behandelt werden.

Es kann zwischen zwei Arten von Classful Qdiscs unterschieden werden. Einmal Qdiscs, die beim Erstellen selbstständig ihre Klassen erzeugen und einmal Qdiscs, bei den der Benutzer dies mithilfe von *tc class* selbst machen muss. Für das Klassifizieren der Pakete in die jeweiligen Klassen gibt es drei verschiedene Möglichkeiten: 1. Mithilfe von Filtern, wobei die Aktion definiert, in welche Klasse das Paket kommt. 2. Anhand des Type of Service (ToS) Feld des IPv4 Headers. 3. Über die SKB-Priorität.

Wie bei denn Classless Qdiscs gibt es auch hier die Möglichkeit, mehrere Classful Qdiscs hintereinander zu schalten. Das bedeutet, es ist möglich, eine verschachtelte Qdisc zu definieren, welche aus Classless und Classful Qdiscs bestehen kann. Vorstellen kann man sich diese wie ein Hierarchie mit mehreren verschiedenen einzelnen Qdiscs wobei die Root-Qdisc den Anfang darstellt. In Abbildung 2.4 ist ein Beispiel einer verschachtelten Qdisc dargestellt. Damit solch eine definiert werden kann, werden IDs benötigt. Diese setzen sich aus einer Major- und einer Minor-Zahl zusammen, welche jeweils 16 Bit groß sind. Geschrieben wird die ID dann Major:Minor wobei Major und Minor als Hexadezimalzahl angegeben werden. Es gibt zwei spezielle IDs: einmal die ID, bei der alle Bits auf 1 gesetzt sind, also ffff:ffff, welche für Root steht und einmal die ID, bei der alle Bits auf 0 gesetzt sind, welche für unspezifiziert steht. Jede Qdisc muss, wie in Abbildung 2.4 zusehen, ein Parent-ID spezifiziert haben. Diese gibt an, an welcher Position beziehungsweise unter

welcher Qdisc diese gesetzt wird. Um eine Qdisc an Root zusetzten kann anstatt der Parent-ID ffff:ffff auch einfach Root geschrieben werden. Eine Ausnahme stellt wieder die ingress/clsact Qdisc dar, denn bei dieser muss kein Parent angegeben werden, weil diese nur an einer bestimmte Position gesetzt werden kann. Dabei wird der Parent automatische auf die speziell dafür definierte ID ffff:fff1 gesetzt. Außerdem können an diesen beiden keine Kinder hinzugefügt werden.

Wie in Abbildung 2.4 zusehen muss an jeder Qdisc, welche Kinder hat ein Handle-ID definiert sein. Dabei wird bei dieser nur die Major-Zahl spezifiziert. Der Handle wird für das Setzten von Kinder an einer Qdisc benötigt, denn dadurch wird die Parent-ID des Kindes teilweise bestimmt. Das bedeutet, die Major-Zahl der Parent-ID des Kindes entspricht dem Handle der Vater-Qdisc. Somit kann die Hierarchie von Qdiscs eindeutig definiert werden.

Wenn eine Klasse zu einer Classful Qdisc hinzugefügt wird, gilt der Zusammenhang zwischen Handle und Parent-ID auch. Bei Klassen ist es aber mögliche, wie in Abbildung 2.4 zusehen, mehrere Klassen an einer Qdisc hierarchisch zuzuordnen. Dabei wird bei der Klasse eine Classid spezifiziert, welche die gleiche Aufgabe hat, wie der Handle bei Qdiscs. Ein Unterschied ist nur, dass hierbei auch die Minor-Zahl angegeben werden muss. Zu beachten ist, wenn eine Hierarchie von Klassen an einer Qdisc definiert wird, dass alle Klassen als Major-Zahl der Parent-ID den spezifizierten Handle dieser Qdisc haben. Wie bei den Qdiscs müssen sich alle Classids in der Minor-Zahl unterscheiden. Wenn hinter eine Klasse eine Qdisc gesetzt wird, wird als Parent-ID die Classid verwendet. Beim Hinzuzufügen von Filtern muss auch eine Parent-ID angegeben werden. Falls der Parent eine Qdisc ist, muss hierbei auch wieder der Zusammenhang zwischen Parent-ID und Handle-ID bestehen. Es ist aber auch möglich, ein Filter an eine Klasse zusetzten. Das geht, indem als Parent-ID des Filters die spezifizierte Classid der Klasse benutzt wird. Wenn diese Filter verwendet werden, um die Pakete zu klassifizieren, wird mit einer Aktion die Flowid spezifiziert. Diese wird dabei auf die entsprechende Classid der Klasse gesetzt, in welche das Paket kommen soll. Beim Erstellen einer verschachtelten Qdisc gibt es keine Einschränkungen, bis auf das manche Kombinationen von Qdiscs nicht unterstützt werden.

Wenn im Linux-Network-Stack die Methode *enqueue()* aufgerufen wird, wird immer *enqueue()* der Root-Qdisc aufgerufen. Falls diese nur eine einfache Classless Qdisc ist, wird das Paket in diese eingefügt. Doch wenn es sich um eine Classful Qdisc handelt, wird das Paket mit der entsprechenden verwendeten Klassifizierung in eine Klasse eingeteilt. Wenn es sich bei der Root-Qdisc um eine verschachtelte Qdisc handelt, durchläuft das Paket jeweils die einzelnen Qdiscs. Dabei ruft die Vater-Qdisc immer *enqueue()* der Kind-Qdisc auf. Dies wird solange gemacht, bis das Paket in der untersten Hierarchieebene angekommen ist. Dort bleibt es, bis es mit *dequeue()* wieder entnommen wird. Wenn *dequeue()* im Linux-Network-Stack aufgerufen wird, beginnt auch dieser Methodenaufruf an der Root-Qdisc. Dabei wird wieder *dequeue()* der entsprechenden Kind-Qdiscs aufgerufen. In welcher Reihenfolge und zu welchem Zeitpunkt die Pakete letztendlich entnommen werden, wird von der gesetzten Qdisc definiert. Da bei einer verschachtelten Qdisc die Pakete entsprechend durch alle einzelnen Qdiscs gehen, werden dementsprechend auch alle Eigenschaften der einzelnen Qdiscs berücksichtigt.

Um eine verschachtelte Qdisc zu löschen, reicht es aus, nur die Root-Qdisc zu löschen. Damit werden auch alle gesetzten Filter und Klassen dieser gelöscht. Beispiele für einfache Classful Qdiscs sind die Hierarchical-Fair-Service-Curve (HFSC) Qdisc [4] und die Hierarchy-Token-Bucket (HTB) Qdisc [5].

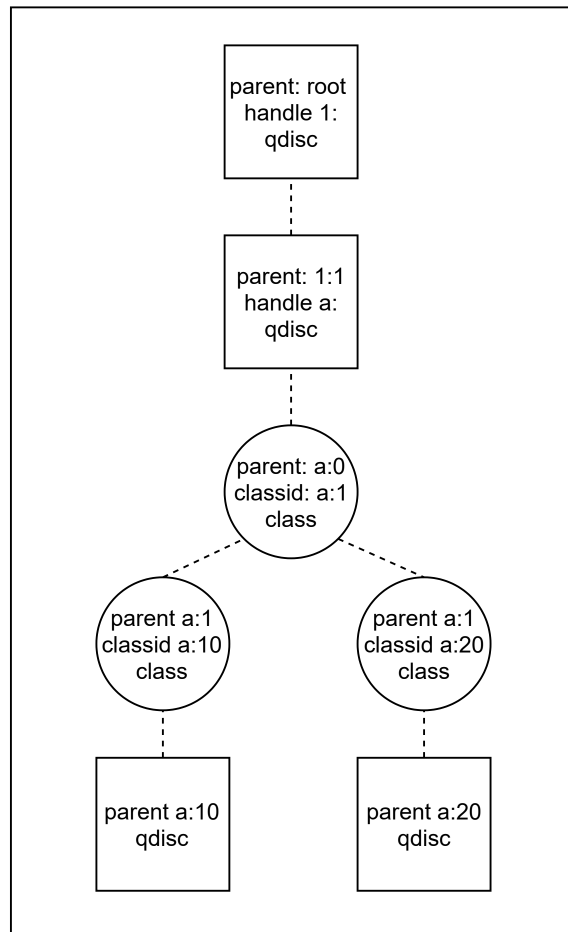


Abbildung 2.4: Beispiel einer verschachtelten Qdisc

TAPRIO Qdisc

Die TAPRIO Qdisc ist eine Linux-Implementierung für einen TAS und ermöglicht somit deterministische Echtzeitkommunikation mithilfe eines zeitgesteuerten Schedulings. Da diese Qdisc ein wichtiges Element in dieser Arbeit darstellt, wird diese hier genauer erklärt. TAPRIO ist eine Classful Qdisc, welche seit Kernel-Version 4.20 im Linux-Kernel integriert ist. Diese kann nur an Interfaces gesetzt werden, welche mindestens zwei TX-Queues haben, also multiqueued sind. Weiter ist es nur möglich, TAPRIO als Root-Qdisc zusetzten. Die Funktionsweise des TAPRIOs hat einen wesentlichen Unterschied zu dem eines TAS. Denn dieser klassifiziert die Pakete anhand der SKB-Priorität.

In Abbildung 2.5 sieht man den internen Aufbau der TAPRIO Qdisc, wenn diese an einem Interface, welches vier TX-Queues hat, gesetzt wird. Die Klassen werden dabei genauso wie die pfifo Qdiscs automatisch erzeugt. Dabei entspricht die Anzahl von Klassen mit pfifo Qdisc immer der Anzahl der TX-Queues des jeweiligen Interfaces auf dem TAPRIO gesetzt ist. Dadurch ist auch klar, warum TAPRIO nur an multiqueued Interfaces gesetzt werden kann. Denn sonst gibt es nur eine Klasse und dementsprechend ist kein priorisiertes Scheduling möglich. Außerdem ist jeder pfifo Qdisc eine TX-Queue zugeordnet, in diese die jeweiligen Pakete dann hinzugefügt werden.

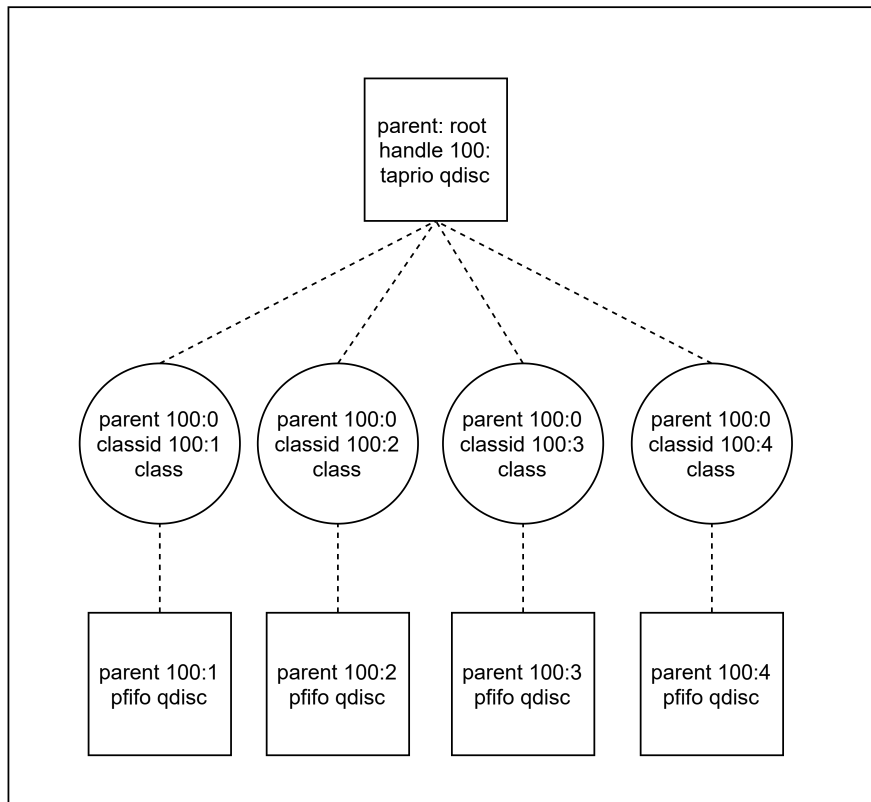


Abbildung 2.5: Interner Aufbau von TAPRIO an Interface mit vier TX-Queues

Wenn mit *enqueue()* ein Paket zu TAPRIO hinzugefügt wird, wird dieses anhand der SKB-Priorität in die entsprechende pfifo Qdisc einsortiert. Beim Aufruf von *dequeue()* werden alle pfifo Qdiscs der Reihe nach durchgegangen. Dabei wird jeweils das nächste Paket der aktuellen pfifo Qdisc angeschaut und wenn das dazugehörige Gate gerade offen ist, wird das Paket zurückgegeben und aus der Qdisc entfernt. Wenn das Gate zu ist, wird mit der nächsten pfifo Qdisc weitergemacht. Falls kein Paket entnommen werden kann, wird *Null* zurückgegeben. Diese führt dazu, dass die in Listing 2.1 ausgeführte Schleife abgebrochen wird. Wenn man jetzt von dem Fall ausgeht, das keine neuen Paket über dieses Interfaces gesendet werden, wird dementsprechend auch *dequeue()* nicht mehr aufgerufen. Dies hat zur Folge, dass keine Pakete mehr gesendet werden, obwohl noch welche gesendet werden müssten. Das darf aber im Hinblick auf die harten Echtzeitgarantienen nicht passieren. Deshalb gibt es einen Timer, welcher beim Erzeugen von TAPRIO erstellt wird. Dieser löst immer nach einer bestimmten Zeit, welche durch die GCL Einträge gegeben ist, den *NET_TX_SOFTIRQ* Interrupt aus. Dieser stößt letztendlich ein *dequeue()* an. Für eine genauere Erklärung siehe Abschnitt 2.2. Somit werden alle Pakete rechtzeitig gesendet und die Echtzeitgarantien können eingehalten werden.

In Listing 2.2 sieht man den tc-Befehl zum Hinzufügen der TAPRIO Qdisc auf dem Interface veth0 als Root-Qdisc. Der Parameter *num_tc* gibt die Anzahl der TC an. TAPRIO unterstützt bis zu 16 verschiedene TC, dafür werden aber 16 TX-Queues benötigt, weil es für jede TC mindestens eine Queue geben muss. Mit *map* wird das Mapping von SKB-Priorität auf TC mithilfe des Bit-Vektors angegeben. Dabei entspricht das erste Bit von links der Priorität 0 welche im Beispiel auf TC1 gemappt wird. Das zweite Bit von links mappt die Priorität 1 auf TC0. Der Rest erfolgt anaolg. Mit dem Parameter *queues* wird das Mapping von TC auf Queues angegeben. Der erste Eintrag von links

Listing 2.2 Beispiel tc Add TAPRIO Befehl

```
tc qdisc add dev veth0 parent Root handle 100: taprio \
        num_tc 2 \
        map 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 \
        queues 1@0 1@1 \
        base-time 1554445635681310809  \
        cycle-time 1000000
        sched-entry S 01 800000 \
        sched-entry S 02 200000 \
        clockid CLOCK_TAI
```

mappt dabei TC0 auf TX-Queues, der zweite Eintrag mappt TC1 auf TX-Queues usw. Es ist möglich, für eine TC mehrere Queues zu definieren. Dazu werden Bereiche von Queues angegeben mit der Notation queue_count@queue_offset. Dazu stellt man sich ein Array vor, in dem jede Array-Position eine Queue darstellt. Die Bereiche sind durch [queue_offset:queue_count+queue_offset-1] gegeben. Dieser beinhaltet dann die Queues an den entsprechenden Array-Positionen. Die spezifizierten Bereiche dürfen sich nicht überlappen. Dieses Mapping wird im Linux-Network-Stack für die Auswahl der TX-Queues verwendet.

Der Parameter *base-time* gibt die Basiszeit und *cycle-time* gibt die Zykluszeit, also die Zeit, die benötigt wird, einmal die GCL zu durchlaufen, an. Beide Zeiten werden in Nanosekunden angegeben. Der k-te Zyklus beginnt zum Zeitpunkt *base-time* + k**cycle-time*. Falls die *cycle-time* nicht spezifiziert wird, wird die Zykluszeit aus Summe der Intervall-Zeiten berechnet. Diese werden in dem Parameter *sched-entry* angegeben. Der Aufbau dieses Parameters ist folgender: *sched-entry* <Befehl> <Gatemaske> <Intervall>. Zurzeit wird nur der Befehl “S” unterstützt. Dieser bedeutet das Setzen des Gatezustandes. Die Gatemaske wird als Hexadezimalzahl angegeben. Dabei entspricht das 0-te Bit (Least Significant Bit) der TC0 und die restlichen Bits analog der jeweiligen TC. Wenn ein Bit in der Gatemaske gesetzt ist, bedeutet dies, dass das entsprechende Gate offen ist. Durch setzten mehrerer Bits ist es möglich, mehrere Gates gleichzeitig offen zu haben. Das Intervall spezifiziert in Nanosekunden, wie lange dieser Zustand bleibt, bis zum nächsten Eintrag gegangen wird. Alle Einträge mit dem Parameter *sched-entry* spezifizieren die GCL. Mit dem Parameter *clockid* wird die Referenz-Uhr angegeben, welche von TAPRIO genutzt werden soll. Es können verschiedenen Uhren verwendet werden wie zum Beispiel CLOCK_TAI, CLOCK_REALTIME oder CLOCK_MONOTONIC. [13][1]

2.4 ip Command-Line-Programm

Mit dem ip-Tool können verschiedene Netzwerkelemente erstellt und konfiguriert werden. Die Funktionsweise von Mininet baut auf zwei von diesen Elementen auf. Diese sind das Virtual Ethernet Interface (veth) und die Network-Namespace (netns). Deshalb werden diese beiden Elemente nachfolgend erläutert. Das ip-Programm befindet sich im Package iproute2. Mit diesem kann das Verhalten des Netzwerkes über das Terminal kontrolliert werden. Zum Beispiel indem die Routing-Tabellen des Kernels verändert oder neue Netzwerkschnittstellen, Interfaces oder Links/Tunnels angelegt werden. Wie tc verwendet auch diese Programm Netlink für die Kommunikation in den Kernel-Space.

2.4.1 Network-Namespace

Das Konzept von Namespaces ermöglicht die Partitionierung des Linux-Kernel in verschiedene Bereiche, welche unabhängig voneinander sind. Die netns sind spezielle Namespaces für das Networking. Dabei sind alle netns isoliert voneinander und jeder netns hat logisch gesehen ein eigenen Linux-Network-Stack. Außerdem besitzt jeder netns eigene Ressourcen. Das bedeutet, dass jeder netns nur eine Menge von IP-Adressen kennt, also nur die IP-Adressen der Interfaces, welche sich in diesem netns befinden. Jeder netns besitzt außerdem eigene Routing- und Address-Resolution-Protocol (ARP)-Tabellen sowie Firewall-Regeln. Der Root-Namespace ist immer vorhanden und stellt den Standard-Namespace dar. Angelegt werden können diese mit dem Befehl *ip netns add NAME*, wobei *NAME* ein global eindeutiger Bezeichner für dieses netns darstellt. Dieser wird benötigt, um zum Beispiel Befehle in diesem ausführen zu können. Für weitere Informationen und Befehle siehe [11].

2.4.2 Veth

Das veth [6] ist ein virtuelles Interface, welches sich wie ein normales physisches Interface verhält. In Listing 2.3 ist ein Beispiel-Befehl zu sehen, mit dem ein veth erzeugt wird. Dabei sind veth0 und veth1 die Namen der beiden erzeugten Interfaces. Wie an diesem Befehl zu erkennen ist, kann ein veth immer nur als Paar erstellt werden, wobei die jeweiligen erzeugten Interfaces dann miteinander verbunden sind. Das bedeutet, wenn ein Paket über ein Interface des Paares gesendet wird, wird es sofort auf dem anderen empfangen. Realisiert wird dies, wenn die Methode zum Senden aufgerufen wird, wird das zusendende Paket direkt in die RX-Queue des anderen Interface des Paares eingefügt. Wenn ein veth-Paar erzeugt wird, haben jeweils beide Interfaces standardmäßig nur eine TX- und eine RX-Queue.

Listing 2.3 Beispiel ip Add Veth Befehl

```
ip link add name veth0 type veth peer name veth1
```

Eine wichtige Anwendung von veth-Paaren ist die Verbindung von netns. In Abbildung 2.6 sind vier netns dargestellt. Um die Kommunikation zwischen den netns zu ermöglichen, werden veth-Paare verwendet. Dazu befindet sich, wie in Abbildung 2.6 zusehen ist jeweils ein Interface des veth-Paares in den beiden netns, welche miteinander verbunden werden sollen. Damit kann über das veth zwischen den beiden netns kommuniziert werden. Außerdem kann man an diesem Kontext auch gut den Sinn von Software-Switches sehen. Denn ohne einen Switch würden man in diesem Beispiel drei weitere veth-Paare benötigen, damit jeder netns mit jedem anderen kommunizieren könnte. Denn weil der Switch Pakete entsprechend weiterleitet, können auch zwei netns miteinander kommunizieren, welche nicht direkt über ein veth-Paar verbunden sind.

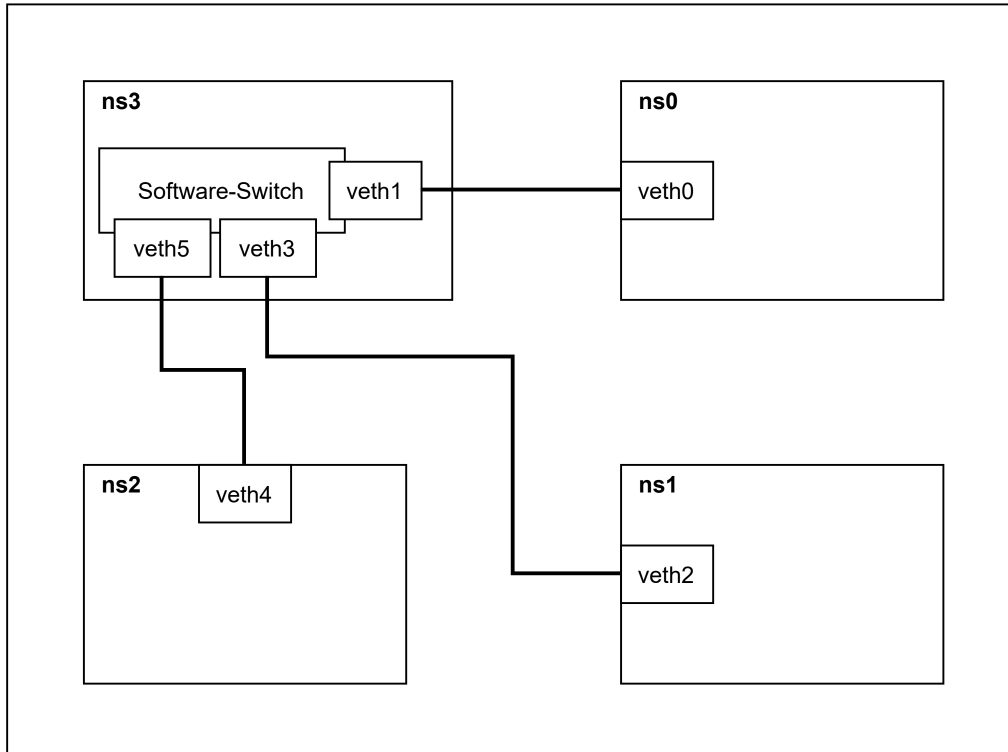


Abbildung 2.6: Network-Namespaces mit veth und Software-Switch

2.5 Mininet

Mininet ist ein bekannter Open-Source-Netzwerk-Emulator für SDN-Netze, welcher hauptsächlich in Python geschrieben ist. Mininet kann aus dem offizielle Git-Repository [9] auf Github heruntergeladen werden. Die verwendete Version in dieser Arbeit ist Version 2.3.0. Mit diesem Emulator können sehr große virtuelle Netzwerke, also Netzwerke mit vielen Switches und Hosts, auf einem Computersystem emuliert werden.

2.5.1 Funktionsweise

In Abbildung 2.7 ist der Netzwerkaufbau von Mininet mit einer einfachen Netzwerktopologie, welche aus zwei Hosts (h1, h2) und einem Switch (s1) besteht, zu sehen. Anhand dieses Aufbaues wird im Folgenden die Funktionsweise von Mininet erklärt. Wie in Abbildung 2.7 zu sehen, werden alle Hosts jeweils als ein Shell-Prozess dargestellt. Damit sich diese Prozesse wie vernetzte Geräte verhalten, wird auf das Konzept der netns zurückgegriffen. Dabei wird für jeden Host ein eigener netns angelegt, und somit sind alle Hosts im Kernel voneinander getrennt. Mininet nutzt hier sogenannte anonyme oder unbenannte netns. Das bedeutet, dass der Shell-Prozess in einem netns gestartet wird und der netns ausschließlich für diesen Prozess angelegt wird. Dementsprechend hat der netns auch keinen Namen/Bezeichner, weil auf diesen außerhalb von Mininet nicht zugegriffen werden soll.

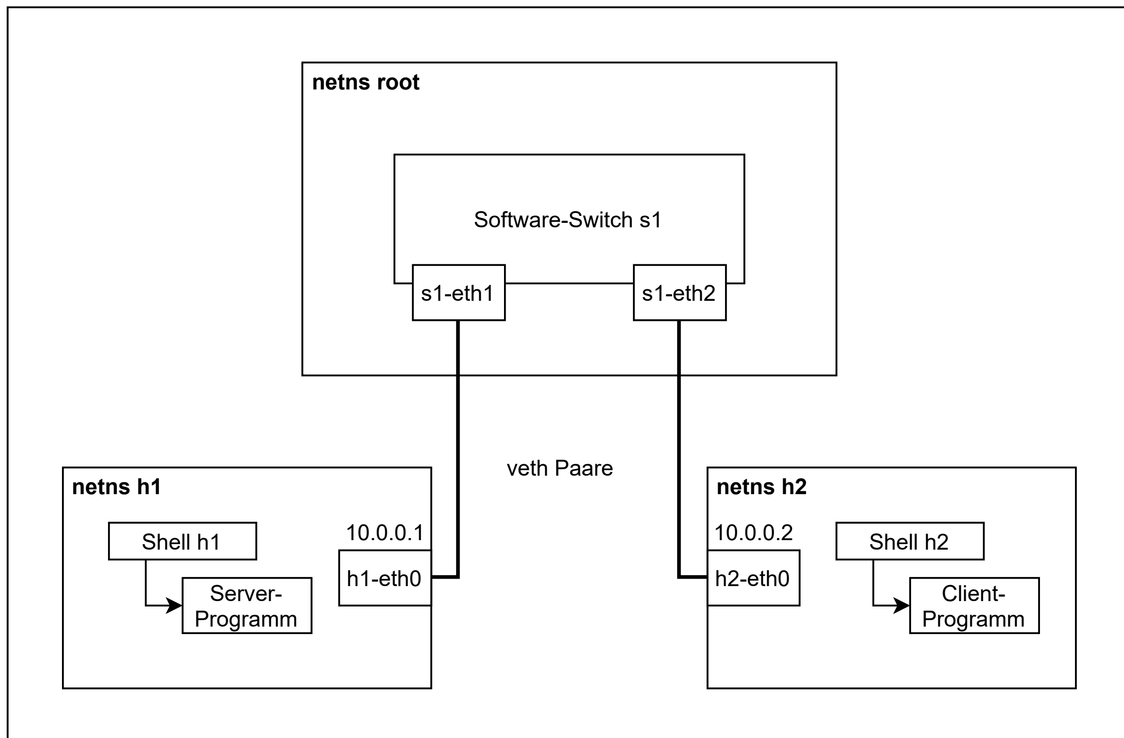


Abbildung 2.7: Beispiel Netzwerkaufbau mit Mininet

Auf den Hosts, also in der jeweiligen Shell, kann ein Programm zum Beispiel ein Client oder Server ausgeführt werden, welches sich so verhält, als wäre es auf einem anderen Rechner. In Abbildung 2.7 ist das schematisch dargestellt, indem in der Shell von Host h1 ein Server-Programm und in der Shell von h2 ein Client-Programm läuft. Damit die Hosts miteinander kommunizieren können, müssen die verschiedenen netns miteinander verbunden werden. Dazu werden veth-Paare verwendet, welche einen virtuellen Link zwischen den netns darstellen. Dabei ist jeweils ein Interface des veth-Paares in den beiden netns, welche verbunden werden sollen. Die Interfaces, welche zu einem Host gehören, also im jeweiligen netns sind, haben jeweils die IP-Adresse des Hosts. Als Switches werden Software-Switches verwendet. Es können verschiedene Software-Switches verwendet werden, wie zum Beispiel die Linux-Bridge oder der Open vSwitch (OVS). Der OVS ist ein SDN-Software-Switch und ermöglicht somit das SDN-Feature, um die Forwarding-Tabellen zu definieren. Das bedeutetet der Benutzer kann dieses Tabellen mit Einträgen füllen und damit definieren, wie der Switch Pakete weiterleiten soll. Mininet verwendet als Standard-Switch den OVS Kernel-Switch. Die Switches werden auch als Prozesse dargestellt und laufen standardmäßig alle im Root-Namespace. Die jeweiligen Interface der veth-Paare, welche an Switches angeschlossen sind, haben keine IP-Adressen. Somit sind alle Hosts jeweils in einem eigenen netns, und alle Switches befinden sich standardmäßig im Root-Namespace, und die Kommunikation zwischen den Hosts ist nur über die veth-Paare möglich.

Eine wichtige Funktionsweise von Mininet ist noch das Link-Eigenschaften wie Datenrate emuliert werden können. Realisiert werden diese mit verschiedenen Qdiscs, welche als Sequenz an das entsprechende Interface gesetzt werden. Dazu gibt es die spezielle Klasse TCIntf, welches gerade diese Qdiscs entsprechend konfiguriert. Für die Begrenzung der Datenrate kann zwischen drei

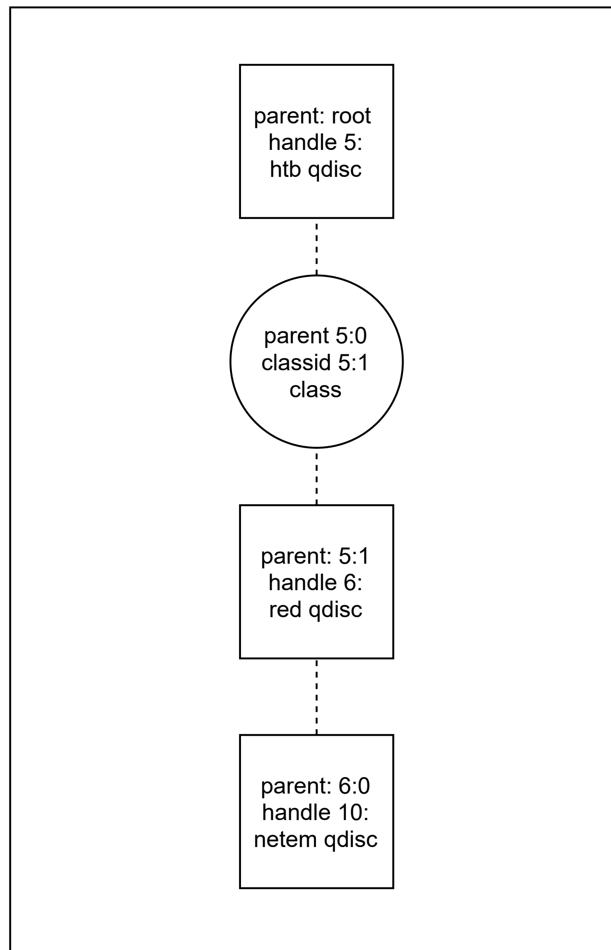


Abbildung 2.8: Beispiel Kombination der Mininet-Qdisc

verschiedenen Qdiscs gewählt werden. Das sind die HFSC, tbf oder HTB Qdisc. Die Datenrate wird in MBit/s angegeben. Wenn die tbf Qdisc verwendet wird, kann zusätzlich noch die Latency in ms angegeben werden. Diese gibt an wie lange sich Pakete höchstens in der tbf Qdisc befinden dürfen. Für einen Überlastvermeidung wird die red Qdisc verwendet. Diese kann in zwei vordefinierten Varianten einmal mit Explicit Congestion Notification (ECN) und einmal ohne verwendet werden. Mithilfe der netem Qdisc kann die Link-Verzögerung in ms, den jitter in ms und den Paketverlust in Prozent spezifiziert werden. Außerdem ist es möglich, die maximale Größe der netem Qdisc anzugeben. Diese gibt an, wie viele Pakete sich maximal gleichzeitig in dieser befinden dürfen. In Abbildung 2.8 ist eine mögliche Kombination von Qdiscs dargestellt, wie diese in Mininet verwendet werden kann. Der Aufbau dieser Sequenz wird durch die Mininet entsprechend erzeugt. Die erste Qdisc ist für die Datenrate zuständig, gefolgt von der red Qdisc und als letztes kommt noch die netem Qdisc. Es muss nicht immer jede Qdisc verwendet werden. Dementsprechend kann auch nur die Datenrate begrenzt oder zum Beispiel der Paketverlust definiert werden. Im Folgenden wird diese Kombination von gesetzten Qdiscs als Mininet-Qdisc bezeichnet.

2.5.2 Benutzung

Um Mininet zu starten, muss der Befehl *sudo mn* im Terminal ausgeführt werden. Dadurch wird die Standard-Netzwerktopologie, welche in Abbildung 2.7 zu sehen ist, erzeugt. Außerdem wird das Mininet-Command-Line-Interface (CLI) gestartet mit diesem kann der Benutzer mit Mininet interagieren. Der Befehl *sudo mn* kann um Argumente erweitert werden. Zum Beispiel kann durch *--switch* ein Switch spezifiziert werden, welcher genutzt werden soll. Außerdem kann mit *--topo* die erzeugte Netzwerktopologie geändert werde. Mininet unterstützt ein Paar vordefinierte Topologien wie zum Beispiel Linear oder Tree. Es gibt auch die Möglichkeit, mit *--costum* eine eigene Topologien zu laden. Weitere Informationen über alle Argumente und entsprechende Hinweise können mit *sudo mn --help* erhalten werden. Im Mininet-CLI können Befehle innerhalb von Mininet ausgeführt werden. Zum Beispiel der Befehl *net*, mit diesem die aktuellen Netzwerkverbindungen angezeigt werden können. Oder *pingall*, welcher ein Ping zwischen allen Host Paaren durchgeführt, womit getestet werden kann, ob alle Hosts erreichbar sind. Außerdem ist es möglich, auf einem Host oder Switch etwas auszuführen. Das geht, indem zuerst der Name des Hosts/Switches eingegeben wird, auf dem der Befehl ausgeführt werden soll, gefolgt von dem auszuführenden Befehl. Dabei wird der Befehl in der jeweiligen Shell ausgeführt. Es gibt noch weitere Befehle, welche mit *help* als Liste angezeigt werden.

Damit Mininet mit eigene Topologien gestartet werden kann, gibt es die Mininet Python API mit dieser können beliebige Topologien erzeugt werden. Außerdem ist *mn* auch mit dieser API geschrieben. Somit kann mit dieser API nicht nur Topologien erstellt werden, sondern auch eigene Python-Skripte zum Starten von Mininet erstellt werden. Welche dann verwendet werden können, um Experimente/Messungen mit Mininet automatisieren zu können. Dabei startet das Skript zuerst Mininet mit einer gegebenen Topologie. Danach können sofort die eigenen Tests automatische ausgeführt werden. Wenn alle Test fertig sind, wird Mininet beendet und damit die virtuelle Testumgebung. Die API bietet viele Möglichkeiten für eigene Konfigurationen von Hosts, Switches und Links. Es kann zum Beispiel wie beim *mn* Skript auch die verwendeten Switches geändert werden. Außerdem können die verwendeten Links geändert werden, um entsprechende Eigenschaften zu erreichen. Dazu müssen die entsprechenden Werte dafür angegeben werden und wenn die Datenrate begrenzt werden soll, muss außerdem spezifiziert werden, mit welcher Qdisc dies realisiert werden soll.

3 Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten in diesem Themenfeld erläutert und eine Abgrenzung zu dieser Arbeit vorgenommen.

Ein Ansatz zum Analysieren von TSN-Netzwerken sind die ereignisorientierten Netzwerk-Simulatoren (engl. DES). Dabei wird der Zeitpunkt, wann ein Ereignis ausgeführt werden soll, vorausberechnet, und die Ereignisse werden in einer Liste nach dem Ausführungszeitpunkt sortiert gespeichert. Somit wird nicht dauerhaft simuliert, sondern nur an Zeitpunkten, an denen Ereignisse auftreten, wie beispielsweise der Empfang oder das Senden eines Paketes. Ein bekanntes Open Source Framework dafür ist das Objective Modular Network Testbed (OMNeT++). Jonathan Falk at al. [2] haben mithilfe von OMNeT++ eine Simulation für TSN-Netzwerke namens NeSTiNg entwickelt. Betrachtet werden dabei zwei verschiedene TSN-Shaping-Arten. Zum einen der CBS und zum anderen der TAS. Ein wesentlicher Nachteil von Simulatoren ist, dass nicht die reale Anwendung verwendet werden kann. Deshalb muss ein Simulator ein abstrahiertes Simulationsmodell verwenden. Doch dadurch ist nicht unbedingt klar, wie sich das reale System im Vergleich zur Simulation verhält.

Jona Herrmann at al. [3] haben einen Time-sensitive/Software-defined Networking (TSSDN) Linux-Software-Switch entwickelt. Dabei wurde als TSN-Komponente der TAPRIO verwendet. Damit dieser in Kombination mit dem Software-Switch funktioniert, muss ein Mapping zwischen dem PCP-Wert und der SKB-Priorität erfolgen. Denn wie schon oben erklärt, nimmt der TAPRIO anhand der SKB-Priorität die Klassifizierung vor, die Pakete aber ihre Priorität im PCP-Feld haben. Realisiert wurde das Mapping mithilfe von acht tc-Filtern. Für jeden PCP-Wert gibt es ein Filter, der als Aktion die SKB-Priorität auf den entsprechenden Wert setzt. Diese Filter werden an der clsact Qdisc gesetzt. Somit können die Filter an Ingress oder Egress gesetzt werden, wobei TAPRIO in Kombination mit beiden Möglichkeiten funktioniert. Evaluiert wurde der entstandenen Software-Switch in Kombination mit einem physischem Interface. Der Fokus dieser Arbeit lag auf der Integration von TAPRIO in Linux-Software-Switches, um diese direkt als Ersatz für Hardware-TSN-Switches zu verwenden. Im Gegensatz dazu ist das Ziel dieser Arbeit, TAPRIO in den Mininet-Netzwerk-Emulator zu integrieren, um damit komplette TSN-Netze auf einem Rechner emulieren zu können.

Gagan Nandha Kumar at al. [8] wollten Source Routing mit TSN verbinden und explizite Forwarding Pfade angelegen. Um TSN zu ermöglichen, wurde der TAPRIO verwendet. Getestet wurde dabei auf einem physischem und virtuellen Interface. Die Messungen mit virtuellen Interfaces wurden in Mininet, also mit dem veth in Kombination mit dem TAPRIO, durchgeführt. Diese Arbeit beschreibt zwar die grundsätzlichen Schritte zur Integration von TAPRIO in Mininet, geht aber nicht im Detail auf die konkrete Umsetzung oder den Entwurf eines entsprechenden Systems ein.

4 Problemstellung

In diesem Kapitel wird ein Überblick über die Probleme gegeben, welche in dieser Arbeit gelöst werden müssen, damit TAPRIO in Mininet genutzt werden kann.

In Abbildung 4.1 ist ein exemplarisches TSN-Netzwerk dargestellt, wie es mit Mininet in Kombination mit TAPRIO erstellt werden soll. Es sollen auch komplexere Netze erzeugt werden können. Doch bei diesen treten keine weiteren Anforderungen auf. Somit reicht es, die Probleme anhand des Netzwerkes in Abbildung 4.1 anzuschauen. In Abbildung 4.1 sieht man den Aufbau einer Topologie mit zwei Hosts und einem Switch, wie diese mit Mininet erzeugt wird. Die beiden Hosts befinden sich entsprechend in einem eigenen netns. Der Software-Switch verbindet die zwei emulierte Hosts jeweils über einen virtuellen Link. Außerdem ist die TAPRIO Qdisc an jedem Interface dargestellt, um das TSN-Scheduling zu ermöglichen. Die Mininet-Qdisc, welche für die Emulation der Link-Eigenschaften wie Link-Verzögerung und Datenrate benötigt wird, ist auch an jedem Interface dargestellt. Denn nur wenn beide Qdiscs gesetzt werden, können reale TSN-Netze emuliert werden.

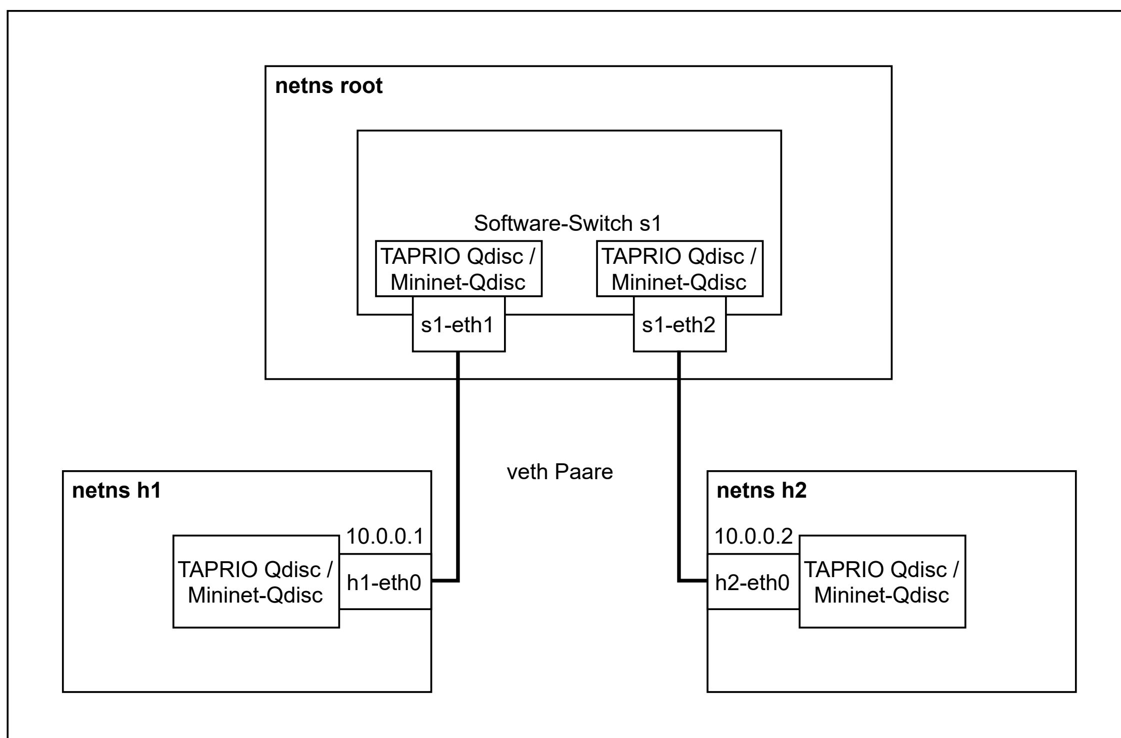


Abbildung 4.1: Mininet Aufbau mit TAPRIO

Das erste Problem besteht darin, dass die TAPRIO Qdisc nur an Interfaces gesetzt werden kann, welche mehrere TX-Queues haben, also multiqueued sind. Aber die verwendeten Interfaces von Mininet, die veth-Paare, haben standardmäßig beim Erzeugen nur eine TX-Queue, sind also singlequeued. Deshalb muss die Möglichkeit geschaffen werden, ein veth mit mehreren TX-Queues zu erzeugen. Damit dann TAPRIO an den Interfaces des veth-Paares gesetzt werden kann.

Ein weiteres Problem ist die Umsetzung von Prioritäten, welche von TAPRIO für das Scheduling benötigt werden. Denn TAPRIO erwartet diese im entsprechenden Feld der SKB-Datenstruktur. Der priorisierte Paketverkehr aber mithilfe des PCP-Wertes erfolgt. Deshalb müssen die von Jona Herrmann at al. [3] entwickelten Filter, welche das Mapping von PCP-Wert auf SKB-Priorität realisieren, verwendet werden. Diese Filter ermöglichen dann die korrekte Funktionsweise von TAPRIO. Damit diese durch den Benutzer einfach gesetzt und gelöscht werden können, wird das Mininet-CLI um entsprechende Befehle ergänzt.

Wenn diese beiden Probleme gelöst wurden sind, kann der TAPRIO in Mininet genutzt werden. Doch damit steht noch nicht die volle Funktionalität von Mininet zur Verfügung. Denn Mininet bietet die Möglichkeit, verschiedene Link-Eigenschaften zu emulieren. Dafür wird, wie in Abschnitt 2.5.1 erklärt, die Mininet-Qdisc verwendet. Doch da es sich bei TAPRIO auch um eine Qdisc handelt, muss eine Lösung gefunden werden, wie beide in Kombination funktionieren. Somit muss es möglich sein, wie in Abbildung 4.1 dargestellt, an einem Interface als Egress Qdisc eine Kombination von beiden zu setzten. Um damit sowohl die Link-Eigenschaften als auch das TSN-Scheduling des TAS emulieren zu können. Da durch diese beiden Elemente die Effekte eines realen Netzes nachgebildet werden sollen, gibt es eine wichtige Anforderung an die Kombination. Denn im realen Netzwerk erfolgt zuerst das Scheduling und danach werden die Pakete über den Link übermittelt, wobei da die Verzögerung entsprechend des Transmission- und Propagation-Delays auftritt. Doch diese Anforderung stellt eine Herausforderung dar, da TAPRIO keine Kombination mit anderen Qdiscs zulässt.

5 Design

In diesem Kapitel werden die Konzepte und der Entwurf eines Systems dargestellt, das die TAS- und Mininet-Funktion integriert und somit die Emulation von TSN durch Mininet ermöglicht.

5.1 Erweiterung Veth um mehrere TX-Queues

Um den TAPRIO an einem Interface setzten zu können, muss dieses mehrere TX-Queues haben. Da aber Mininet das veth, welches standardmäßig nur eine Queue hat, verwendet, muss eine Möglichkeit geschaffen werden, dass ein veth mit mehreren TX-Queues erzeugen werden kann. Dafür bietet das ip-Tool schon eine Lösung, denn beim Hinzufügen eines veth-Paares kann mit dem Parameter *numtxqueues* die entsprechende Anzahl an TX-Queues angegeben werden. Wie in Listing 5.1 zusehen ist, wird dieser Parameter jeweils nach dem Interfacenamen angegeben und somit kann die Anzahl für beide Interfaces des Paares individuell anzugeben werden. Da im Standard IEEE 802.1Q zwischen acht verschiedenen TCs unterschieden wird, wird die Anzahl von TX-Queues in dieser Arbeit standardmäßig auf acht gesetzt. Doch falls einmal mehr als acht TC benötigt werden, insbesondere weil TAPRIO bis zu 16 unterstützt, kann die Anzahl an TX-Queues dann entsprechend angepasst werden.

Listing 5.1 Add veth ip-Befehl mit Parameter numtxqueues

```
ip link add name veth0 numtxqueues 8 type veth peer name veth1 numtxqueues 8
```

5.2 Ermöglichen der Funktion von TAPRIO mit Veth

Damit TAPRIO in Kombination mit dem veth funktioniert, muss ein Mapping von PCP-Wert auf SKB-Priorität erfolgen. Denn TAPRIO benutzt für die Klassifizierung der Pakete die SKB-Priorität. Um dieses Mapping umzusetzen, werden die in [3] entwickelten acht tc-Filter verwendet. Damit diese durch den Benutztер einfach gesetzt und gelöscht werden können, wird das Mininet-CLI um zwei dementsprechende Befehle ergänzt. Die Syntax dieser beiden Befehle ist in Listing 5.2 dargestellt. Mit diesen Befehlen besteht die Möglichkeit, Filter an einem speziellem Interface oder an allen Interfaces von Mininet zu setzten beziehungsweise zu löschen. Wenn der Parameter *all* verwendet wird, werden alle Interfaces berücksichtige bis auf die Loopback (lo) Interfaces. Außerdem muss bei beiden Befehlen immer Ingress oder Egress angegeben werden. Dadurch wird beim add-Befehl spezifiziert, wo die Filter greifen, also ob eingehende oder ausgehende Pakete gefiltert werden und beim del-Befehl wird dadurch angegeben, welche Filter gelöscht werden sollen. Wenn der add-Befehl ausgeführt wird, wird zuerst die für die Filter benötigte clsact Qdisc an das entsprechende Interfaces gesetzt. Danach werden die acht tc-Filter an Ingress oder Egress

Listing 5.2 Hinzugefügte Filter-Befehle für das Mininet-CLI

```
add_filter {dev <Intf name>, all} {ingress, egress}
del_filter {dev <Intf name>, all} {ingress, egress}
show_filter
```

hinzugefügt entsprechend wie es im Befehl angegeben ist. Beim del-Befehl werden alle Filter entweder an Ingress oder Egress gelöscht. Um dem Benutzer eine Übersicht über die gesetzten Filter zu ermöglichen, wird ein weiterer Befehl hinzugefügt. Dieser gibt für alle Interfaces an, ob an diesen jeweils die Filter gesetzt sind. Dabei wird Ingress oder Egress angegeben, wenn die entsprechenden Filter gesetzt sind. Die Syntax dieses Befehls ist auch in Listing 5.2 dargestellt.

Damit TAPRIO in Mininet an jedem Interface funktioniert, muss beim Setzten der Filter etwas beachtet werden. Dabei müssen zwei Fällen unterschieden werden. Beim ersten Fall geht es um ein Interface, das zu einem Host gehört. Wenn an diesem TAPRIO funktionieren soll, müssen dafür an diesem die Egress Filter gesetzt werden. Denn nur dadurch wird bei den Paketen das Mapping vorgenommen. Der andere Fall ist, wenn es sich um ein Interface handelt, welches zu einem Switch gehört. An diesen können im Allgemeinen beide Filter-Typen verwendet werden, solange sichergestellt wird das, bevor die Pakete in TAPRIO kommen, das Mapping umgesetzt wurden ist. Jetzt könnte man sich fragen, warum in diesem Fall überhaupt Filter benötigt werden, wenn davon ausgegangen wird, dass schon die Egress Filter am Host-Interface gesetzt wurden sind. Denn somit hätte doch jedes Paket schon die entsprechende SKB-Priorität gesetzt. Der Grund dafür ist, dass sich jeder Host in einem eigene netns befindet und beim Senden des Paketes über das veth-Paar dieses in ein anderen netns kommt, wo ein neuer SKB für dieses Paket angelegt wird. In diesem neuen SKB ist die gesetzte Priorität nicht mehr vorhanden. Somit muss allgemein gesagt in jedem netns das Mapping mithilfe der Filter erfolgen.

5.3 TAPRIO und Mininet-Qdisc

In diesem Abschnitt werden verschiedene Möglichkeiten betrachtet, wie TAPRIO und die Mininet-Qdisc kombiniert werden können. Dabei wird die red Qdisc nicht betrachtet, da in der Industrie hauptsächlich verbindungslose Protokolle wie zum Beispiel User Datagram Protocol (UDP) verwendet werden, weil im Kontext von Echtzeitkommunikation ein Retransmit kein Sinn machen würde. Der Fokus in dieser Arbeit liegt deshalb auf der Integration des TSN-Scheduling in Kombination mit der Emulation der Link-Verzögerung. Die Begrenzung der Datenrate wird aber auch betrachtet.

Eine gute Kombination ist dabei eine, welcher der realen Situation im TSN-Netzwerk entspricht. Diese ist in Abbildung 5.1 dargestellt. Dabei erfolgt das TSN-Scheduling durch den TAS, welcher die Pakete auswählt, welche gesendet werden. Diese werden dann über den Link zum Ziel-Interface übertragen. In Abbildung 5.1 ist außerdem die entsprechende Umsetzung der realen Situation mit Mininet dargestellt. Dabei wird der TAPRIO für das TSN-Scheduling benutzt. Danach wird mit der Mininet-Qdisc die Link-Eigenschaften, wie Datenrate und Verzögerung, emuliert. Anschließend werden die Pakete über das Quell-Interface zum Ziel-Interface übertragen. Dabei sind diese beiden Interfaces Teil eines veth-Paares und somit wird beim Senden eines Paketes dieses sofort auf dem anderen Interface empfangen.

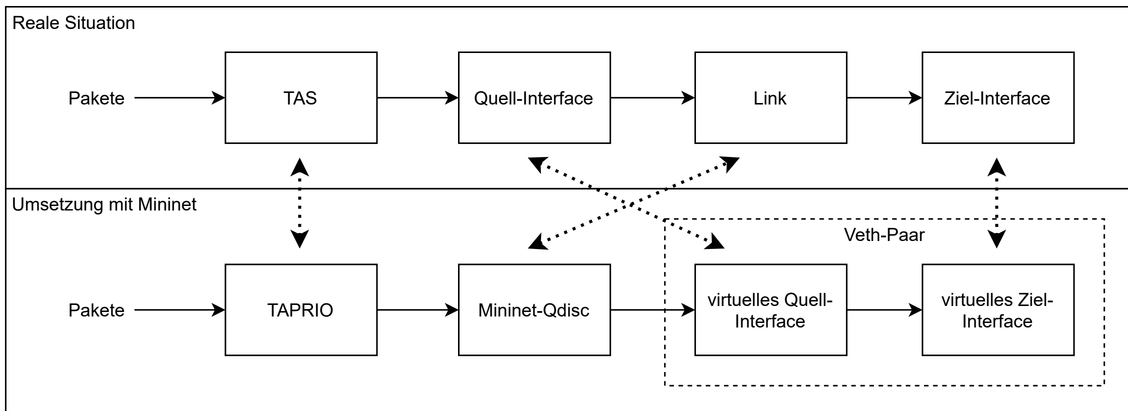


Abbildung 5.1: Reale Situation im TSN-Netzwerk und entsprechende Umsetzung mit Mininet

Es werden aber dennoch alle möglichen Kombinationen, also auch welche, die nicht der Umsetzung in Abbildung 5.1 entsprechen, untersucht, weil bestimmte Anforderungen unter Umständen schwierig umzusetzen sind und deshalb hier ein Kompromiss gefunden werden muss. Im Folgenden werden alle möglichen Kombinationen beispielhaft an einem Interface mit vier TX-Queues veranschaulicht.

5.3.1 Option TAPRIO unter Mininet-Qdisc

Eine erste mögliche Kombination ist in Abbildung 5.2 dargestellt. Hierbei wird unter die Mininet-Qdisc TAPRIO gesetzt. Somit entspricht diese Kombination nicht der Umsetzung in Abbildung 5.1 und dementsprechend auch nicht der realen Situation. Bei dieser Kombination stellt die Einschränkung, dass TAPRIO nur direkt an die Root-Qdisc gesetzt werden kann, ein Problem dar. Um dieses zu lösen, muss das Modul von TAPRIO entsprechend angepasst werden. Wobei dieses Problem sehr einfach gelöst werden kann.

Ein größeres Problem stellt das periodische Scheduling von TAPRIO dar, welches in Abschnitt 2.3.2 genau erklärt wurde. Der dabei ausgelöste *NET_TX_SOFTIRQ* Interrupt hat immer eine Qdisc spezifiziert. Der entsprechende Interrupt-Handler ruft dann *__qdisc_run()* auf der angegebenen Qdisc auf. Diese Methode versucht im Grunde mit *dequeue()* ein Paket aus der Qdisc zu entnehmen und über das Interface zu senden. Im Fall von TAPRIO ist die spezifizierte Qdisc des Interrupts TAPRIO selbst. Somit wird *__qdisc_run()* direkt auf TAPRIO aufgerufen. Außerdem wird beim Senden eines Paketes in der Applikation durch den Linux-Network-Stack *__qdisc_run()* auf der Root-Qdisc, in diesem Fall die Mininet-Qdisc, aufgerufen. Da es sich bei dieser Option, wie in Abbildung 5.2 zusehen ist, um eine verschachtelte Qdisc handelt, wird *dequeue()* entsprechend auf der Kind-Qdisc, also TAPRIO, aufgerufen. Somit kann wegen der Nebenläufigkeit des Kernels *dequeue()* der TAPRIO Qdisc zweimal, einmal durch den Interrupt und einmal durch den Network-Stack, parallel ausgeführt werden. Dies führt dann dazu das der Kernel abstürzt. Außerdem besteht die Möglichkeit, dass beide *dequeue()* Methoden möglicherweise nicht parallel ausgeführt wurden und dementsprechend jeweils ein unterschiedliches Paket entnommen haben. Danach wird versucht beide Pakete gleichzeitig über das gleiche Interface zu senden. Da aber paralleles Senden nicht möglich ist, stürzt der Kernel ab.

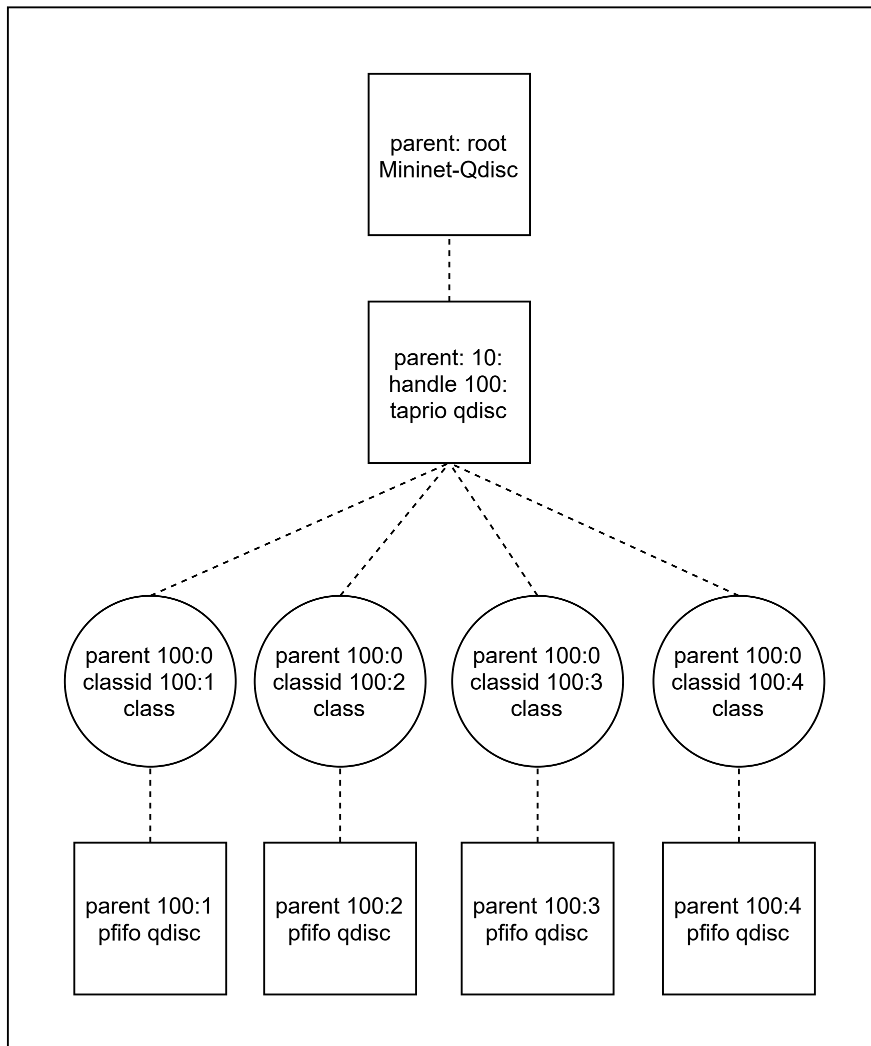


Abbildung 5.2: Option TAPRIO unter Mininet-Qdisc

Der Grund dafür ist, dass der Kritischer Abschnitt (KA), welcher in Listing 2.1 mit einem Spinlock gesichert wurde, bei dieser Option nicht mehr abgesichert ist. Denn dies wird mit einem Lock an der Root-Qdisc erreicht. Da aber in diesem Fall der Interrupt für das periodische Scheduling nicht an der Root-Qdisc greift, sondern direkt an TAPRIO, kann mit diesem Mechanismus der KA nicht mehr gesichert werden. Denn es wird nur die Root-Qdisc durch den Lock gesichert, weil dies für den Linux-Network-Stack ausreichend ist. Dementsprechend werden alle möglichen Kind-Qdiscs nicht durch ein Lock gesichert, weil diese Situation also, dass ein Interrupt nicht an der Root-Qdisc greift, in der Qdisc des Linux-Network-Stacks nicht vorgesehen ist. Somit ist der KA bei dieser Kombination nicht mehr gesichert und der Kernel stürzt ab.

Um dieses Problem zu verhindern, muss TAPRIO entsprechend angepasst werden, damit der Interrupt für das periodische Scheduling auch an der Root-Qdisc greift. Dadurch ist die Absicherung des KA wieder sichergestellt und der Kernel kann nicht mehr abstürzen.

5.3.2 Option TAPRIO vor Mininet-Qdisc

In Abbildung 5.3 ist eine weitere mögliche Kombination dargestellt. Bei dieser wird TAPRIO an Root gesetzt und hinter jede pfifo Qdisc des TAPRIOs wird die Mininet-Qdisc gesetzt. Der Vorteil dieser Kombination ist, dass TAPRIO dabei die Root-Qdisc ist und somit keine Änderungen wie in Abschnitt 5.3.1 vorgenommen werden müssen. Diese Kombination sieht zwar auf den ersten Blick so aus, als würde Sie die reale Situation, wie in Abbildung 5.1 zusehen ist, umsetzten, aber wenn man sich den Ablauf genauer anschaut, stellt man fest, dass es hier einen wesentlichen Unterschied gibt. Um diesen Unterschied klarzumachen, wird daher der tatsächliche resultierende Ablauf hier beschrieben.

Wenn Pakete mit *enqueue()* in TAPRIO eingefügt werden, werden diese entsprechend der SKB-Priorität in die jeweilige pfifo Qdisc einsortiert. Die grundlegende Funktionsweise für verschachtelte Qdiscs definiert, dass *enqueue()* bis auf die unterste Hierarchieebene weiter aufgerufen wird. Das bedeutet, die Pakete werden beim Einfügen direkt in die Mininet-Qdisc kommen. Dementsprechend wird wieder zuerst der Link emuliert und danach erst das Scheduling gemacht und dies entspricht nicht der realen Situation.

Somit ist die Funktionsweise dieser Option die gleiche wie bei der Option bei der TAPRIO nach der Mininet-Qdisc gesetzt ist. Der einzige Unterschied besteht darin, dass bei dieser Option mehrere Miniet-Qdiscs verwendet werden. Realisiert werden kann diese Option aber nicht so einfach, da es nicht möglich ist, unter einer pfifo Qdisc noch weitere Qdiscs zu definieren. Dafür müsste die pfifo Qdisc entsprechend angepasst werden oder eine neue Qdisc implementiert werden, welche im Grunde eine FIFO-Qdisc ist, aber mit der Möglichkeit, Kinder an diese setzten zu können. Daher stellt diese Kombination auch keine gute Lösung dar. Deswegen wird auf das Implementieren dieses Ansatzes in dieser Arbeit verzichtet. Insbesondere da diese Option im Grunde das Gleiche macht wie die Option TAPRIO nach Mininet-Qdisc.

5.3.3 Eigenes Qdisc-Modul

Bisher wurde versucht, nur durch Kombination der TAPRIO und Mininet-Qdisc eine Lösung zu finden, welche der realen Situation entspricht. Doch diese konnte dadurch nicht gefunden werden. Deshalb wird jetzt versucht, mit dem Ansatz, bei dem ein neues Qdisc-Modul definiert wird, solch eine Lösung umzusetzen. Dieses neue Modul muss dabei hauptsächlich das Problem lösen, dass bei einer verschachtelten Qdisc die Pakete mit *enqueue()* bis auf die unterste Hierarchieebene eingefügt werden. Für eine genaue Erläuterung dieses Problems siehe Abschnitt 5.3.2. Um dieses Problem zu Lösen wird die Mininet-Qdisc nicht unter den TAPRIO gesetzt, sondern parallel dazu. Somit werden die Pakete nicht mehr automatisch in die Mininet-Qdisc einsortiert. Dadurch muss aber eine Lösung gefunden werden, wie die Pakete dort eingefügt werden können. Dies wird mit hauptsächlich durch zwei verschiedenen *dequeue()* Methoden gelöst. Eine entnimmt die Pakete aus TAPRIO und fügt diese der Mininet-Qdisc hinzu. Die andere *dequeue()* Methode entnimmt für das Senden die Pakete aus der Mininet-Qdisc. Somit entspricht die Funktionsweise des neuen Moduls der realen Situation. Hierbei werden zwei verschiedene Varianten betrachtet, wie dieses umgesetzt werden kann.

Damit das neue Modul unabhängig von der Mininet-Qdisc ist, ist diese im jeweiligen Aufbau nicht vorhanden, sondern wird durch eine pfifo Qdisc dargestellt. Der Grund, warum diese Unabhängigkeit wichtig ist, ist der das Mininet verschiedene Zusammensetzungen der Mininet-Qdiscs, wie in

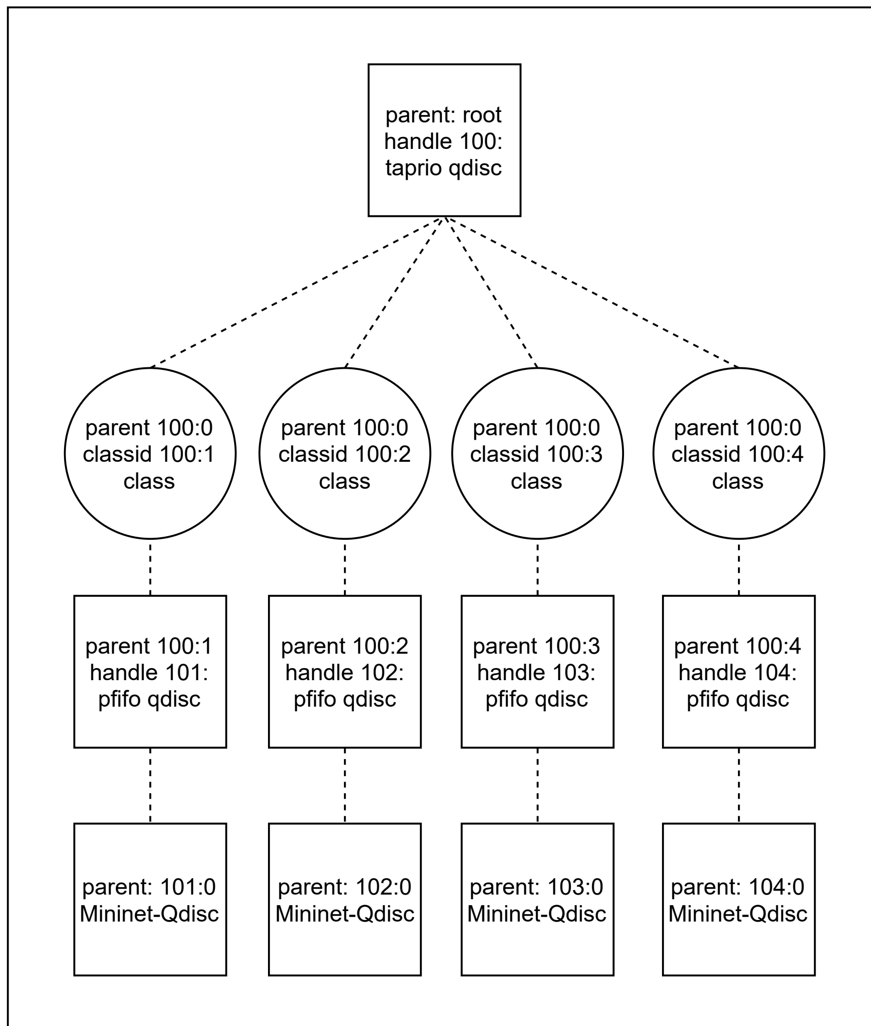


Abbildung 5.3: Option TAPRIO vor Mininet-Qdisc

Abschnitt 2.5.1 beschrieben, unterstützt. Deshalb ist es wichtig, dass das neue Modul unabhängig davon ist, damit weiterhin alle verschiedenen Mininet-Qdiscs verwendet werden können. Diese pfifo Qdisc wird beim Erzeugen der neuen Qdisc benötigt und stellt somit eine Art Platzhalter für die Mininet-Qdisc dar. Um diese pfifo Qdisc von anderen unterscheiden zu können, wird diese im Aufbau immer mit einem dickeren Rahmen markiert. Damit die Mininet-Qdisc gesetzt werden kann, wird der *tc replace*-Befehl benutzt. Dadurch kann die pfifo Qdisc durch die Mininet-Qdisc ersetzt werden.

Um das Hinzufügen und Löschen von TAPRIO für den Benutzer einfach zu machen, wird das Mininet-CLI um entsprechende Befehle ergänzt. Die Syntax dieser ist in Listing 5.3 dargestellt. Beim add-Befehl müssen die Parameter für die TAPRIO Konfiguration angegeben werden. Dabei müssen alle Parameter ab *num_tc* in der gleichen Syntax, wie in Listing 2.2 dargestellt, spezifiziert werden. Der add-Befehl ermöglicht es TAPRIO an einem speziellem oder an allen Interfaces zu setzten. Da Mininet schon beim Starten die Mininet-Qdisc setzt, muss diese beim add-Befehl zuerst gelöscht werden und danach wieder an die entsprechende Position an der neuen Qdisc

Listing 5.3 Hinzugefügte TAPRIO-Befehle für das Mininet-CLI

```
add_taprio {dev <Intf name>, all} <TAPRIO Parameter>
del_taprio {dev <Intf name>, all}
show_taprio
```

gesetzt werden. Wenn TAPRIO gesetzt ist, kann mit diesem add-Befehl auch eine Änderung des Schedulings vorgenommen werden, indem die Parameter entsprechend geändert werden. Beim del-Befehl wird TAPRIO entweder an dem angegebenen Interface gelöscht oder wenn *all* verwendet wird, werden alle gesetzten TAPRIOs gelöscht. Falls die Mininet-Qdisc gesetzt ist, wird diese nach dem Löschen von TAPRIO wieder an Root gesetzt. Bei beiden Befehlen gilt, wenn der Parameter *all* verwendet wird, werden wie bei den Filter Befehlen auch die lo Interfaces nicht berücksichtigt. Um dem Benutzer auch hier eine Übersicht zu liefern, an welchen Interfaces gerade TAPRIO gesetzt ist, wird noch ein show-Befehl ergänzt. Die Syntax dieses Befehls ist auch in Listing 5.3 zu sehen.

Variante TAPRIO_EXTEND Qdisc

Diese Variante beschreibt ein Ansatz bei dem das TAPRIO-Modul um die benötigte Funktionalität erweitert wird. In Abbildung 5.4 ist der entsprechende Aufbau dieser TAPRIO_EXTEND Qdisc dargestellt. Dabei entspricht der Aufbau im Grunde dem Aufbau der TAPRIO Qdisc, wie dieser in Abbildung 2.5 dargestellt ist. Es kommt nur zusätzlich noch eine weitere pfifo Qdisc als Kind-Qdisc hinzu. Diese stellt, wie oben erwähnt, ein Platzhalter für die Mininet-Qdisc dar.

Umgesetzt werden kann diese Variante, indem Pakete mit *enqeueue()* in die ursprünglichen pfifo Qdiscs einsortiert werden. Zum Senden werden Pakete mit *deqeueue()* aus der pfifo Qdisc, welche in Abbildung 5.4 mit einem dickerem Rahmen dargestellt ist, entnommen. Da diese durch die Mininet-Qdisc ersetzt wird, werden die Pakete daraus entnommen und dadurch wird der Link erst nach dem Scheduling emuliert. Jetzt muss nur noch gelöst werden, wie die Pakete von TAPRIO in diese Qdisc kommen. Dafür wird eine zweite *deqeueue()* Methode benötigt. Diese Methode führt in einer Endlosschleife die ursprüngliche *deqeueue()* Methode von TAPRIO aus. Die dadurch entnommenen Pakete werden der Mininet-Qdisc hinzufügen. Somit entspricht die Funktionsweise der TAPRIO_EXTEND Qdisc der realen Situation.

Doch diese Umsetzung hat zwei große Probleme, welche nachfolgend erläutert werden. Das erste Problem ist, dass durch die zwei verschiedenen benötigten *deqeueue()* Methoden der in Abschnitt 2.2 erklärte *NET_TX_SOFTIRQ* Interrupt des Linux-Network-Stacks nicht mehr verwendet werden kann. Denn jede Qdisc hat den *struct Qdisc_ops* definiert, welcher aus Pointer auf die eigenen implementierten Methoden besteht. Somit kann im Network-Stack die allgemeine Methode *deqeueue()*, wie sie im *struct Qdisc_ops* definiert ist, aufgerufen werden und es wird immer die entsprechende Implementierung der jeweiligen Qdisc ausgeführt. Somit kann der Interrupt nur von der *deqeueue()* Methode der TAPRIO_EXTEND Qdisc verwendet werden, da dieser Interrupt möglicherweise im Network-Stack ausgelöst wird und somit entsprechend definiert sein muss. Deshalb gibt es nur die Möglichkeit, *deqeueue()* von TAPRIO in einer Endlosschleife auszuführen, um damit das Scheduling gewährleisten zu können. Das zweite Probleme betrifft das Locking der TAPRIO_EXTEND Qdisc. Denn alle drei Methoden, also *enqeueue()* und *deqeueue()* von TAPRIO_EXTEND sowie *deqeueue()* von TAPRIO, benötigen den gleichen Lock. Dies wird

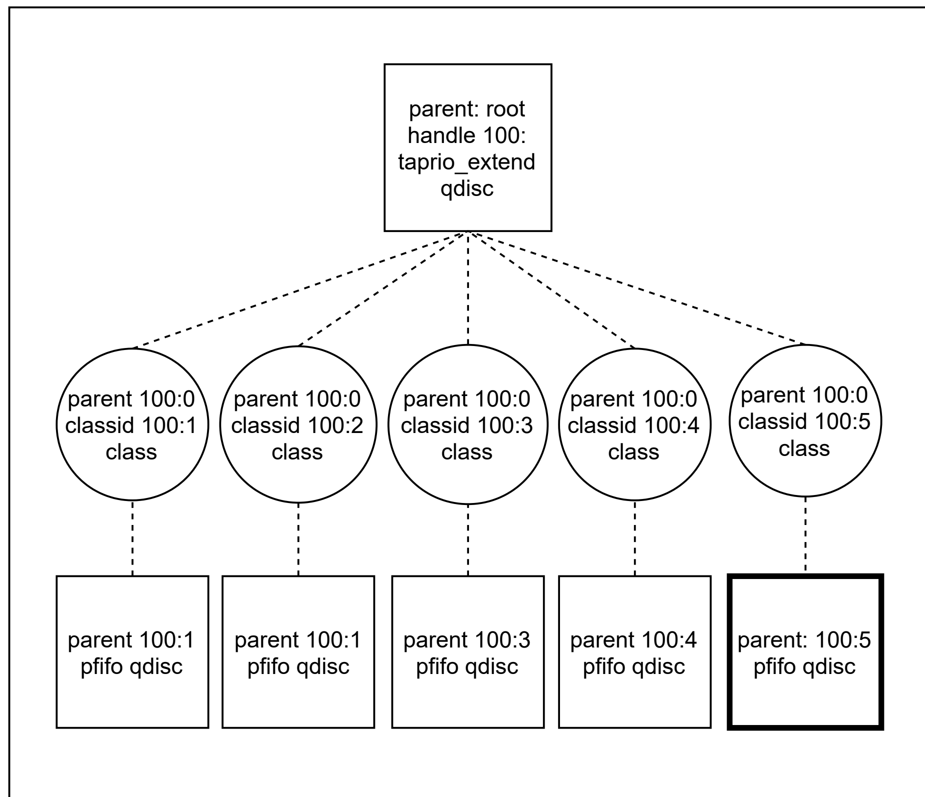


Abbildung 5.4: Aufbau der TAPRIO_EXTEND Qdisc

benötigt, um sicherzustellen, dass alle drei Methoden korrekt funktionieren. Doch dies führt zu Problemen mit der Endlosschleife. Denn diese gibt den Lock dann nicht mehr frei und dadurch können keine Paket mehr in die Qdisc eingefügt werden. Somit ist es mit der Endlosschleife nicht möglich, eine funktionierende Qdisc zu implementieren.

Variante TAPRIO_MININET Qdisc

Da die TAPRIO_EXTEND Qdisc wegen der Endlosschleife nicht realisiert werden kann, wird nach einer Lösung gesucht, welche diese Endlosschleife nicht benötigt. Das bedeutet, es muss eine Lösung gefunden werden, bei der es möglich ist, auch für *deqeueue()* von TAPRIO den *NET_TX_SOFTIRQ* Interrupt verwenden zu können. Dies kann mit der neu entwickelten TAPRIO_MININET Qdisc erreicht werden. In Abbildung 5.5 ist der Aufbau dieser Qdisc dargestellt. Diese besteht aus zwei Kindern, einmal TAPRIO und einmal die pfifo Qdisc als Platzhalter für die Mininet-Qdisc. Die Aufgabe dieser TAPRIO_MININET Qdisc ist nur, die aufgerufenen Methoden an die entsprechende Kind-Qdisc weiterzuleiten. Somit implementiert diese Qdisc nur eine Art Wrapper für TAPRIO und die Mininet-Qdisc. Das bedeutet, dass jeder Methodenaufruf immer an die gleiche Kind-Qdisc geleitet wird. Ansonsten hat die neue Qdisc keine weitere Funktionalität.

Nachfolgend wird die Funktionsweise von TAPRIO_MININET genau erläutert, welche in Abbildung 5.6 schematisch dargestellt ist. Diese beschreibt hauptsächlich, wie die Methodenaufrufe an die entsprechende Kind-Qdiscs weitergeleitet werden. Dabei wird die Funktionsweise anhand

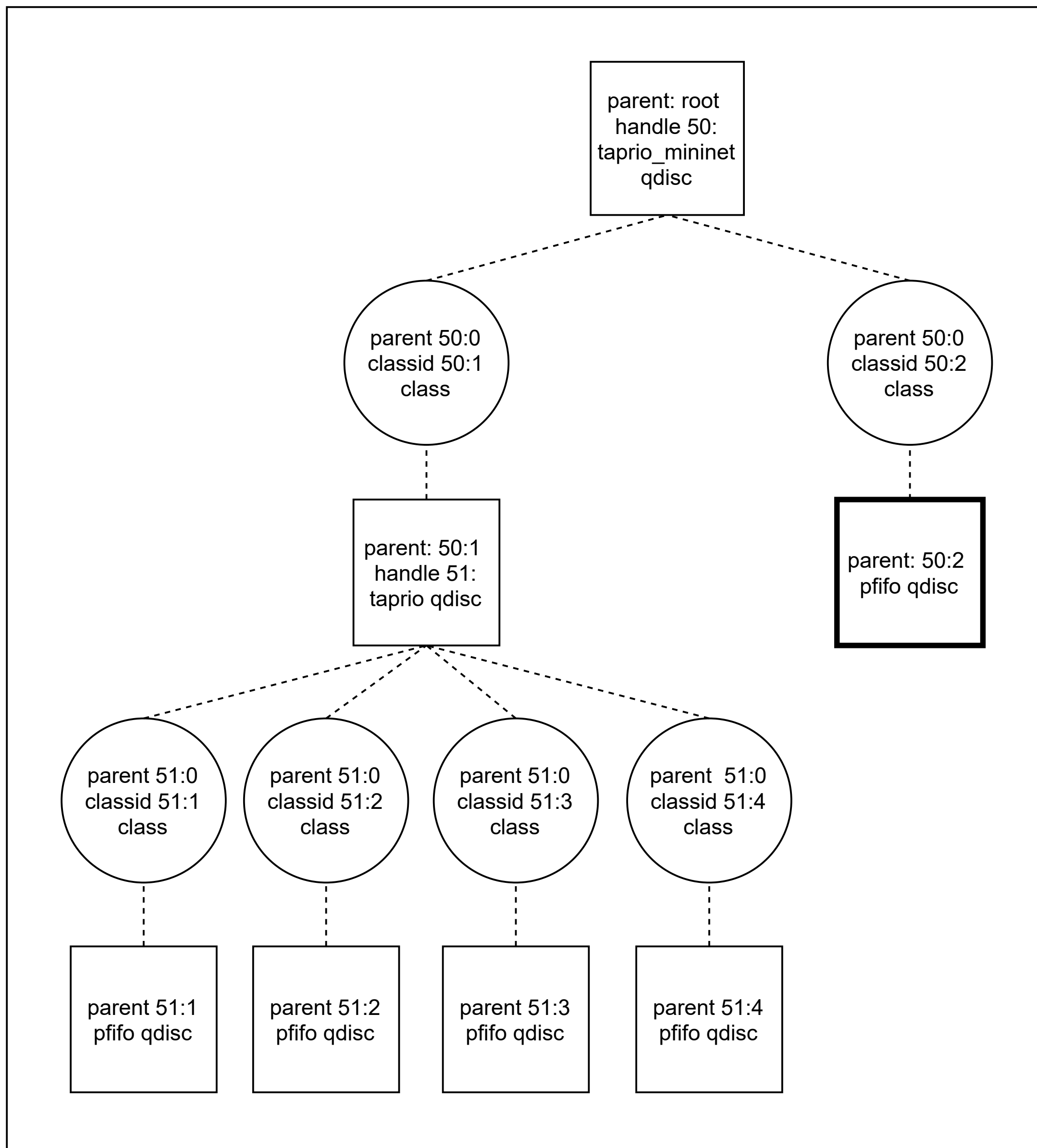


Abbildung 5.5: Aufbau der TAPRIO_MININET Qdisc

eines Paketes erläutert, welches einmal komplett die TAPRIO_MININET Qdisc durchläuft. Das bedeutet, es wird damit angefangen zu erklären, wie ein Paket in TAPRIO_MININET eingefügt wird und die Erklärung endet damit, wenn dieses wieder zum Senden entnommen wurde. Die Funktionsweise wird anhand von Abbildung 5.6 erklärt, wobei die Farben der dargestellten Methoden aufgegriffen werden. Um ein Paket in die TAPRIO_MININET Qdisc einzufügen, wird die rote *enqeueue()* Methode von dieser Qdisc aufgerufen. Diese fügt das Paket mit der entsprechenden *enqeueue()* Methode in die TAPRIO Qdisc ein. Das periodische Scheduling von TAPRIO, wie in Abschnitt 2.3.2 erklärt, ruft mithilfe des *NET_TX_SOFTIRQ* Interrupts die blaue *deqeueue()* Methode auf TAPRIO auf. Die dadurch entnommenen Pakete werden mit der violetten *enqeueue()* Methode in die Mininet-Qdisc eingefügt. Um Pakete für das Senden zu entnehmen, wird die grüne

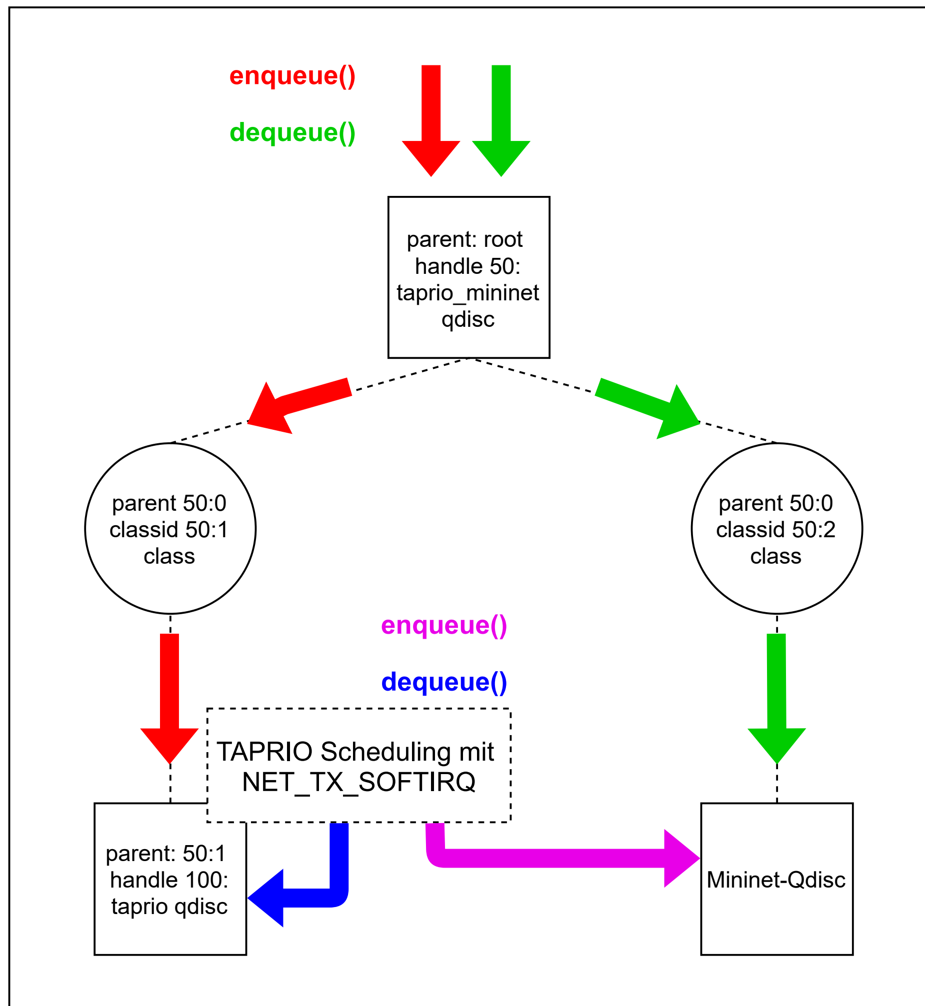


Abbildung 5.6: Funktionsweise der TAPRIO_MININET Qdisc

deqeueue() Methode von TAPRIO_MININET aufgerufen, welche Pakete aus der Mininet-Qdisc entnimmt. Somit entspricht diese Funktionsweise der realen Situation, da zuerst das Scheduling der Pakete durch TAPRIO erfolgt und danach die Link-Eigenschaften durch die Mininet-Qdisc emuliert werden.

5.3.4 TAPRIO_MININET in Kombination mit Datenratenbegrenzung

Wenn TAPRIO_MININET in Kombination mit der Datenratenbegrenzung in der Mininet-Qdisc verwendet wird, funktioniert das TSN-Scheduling nicht mehr korrekt. Der Grund dafür ist, dass die Begrenzung der Datenrate umgesetzt wird, indem Pakete entsprechend verzögert werden. Doch dadurch können die im Scheduling definierten Zeitfenster nicht mehr eingehalten werden, siehe dazu Abschnitt 7.4.6. Um ein korrektes Scheduling in Kombination mit der Datenratenbegrenzung zu ermöglichen, werden verschiedene Möglichkeiten betrachtet, wie dies umgesetzt werden könnte. Ob diese korrekt funktionieren, wird in Abschnitt 7.4.6 evaluiert.

Qdisc zur Datenratenbegrenzung vor TAPRIO_MININET

Eine erste mögliche Umsetzung, wie das Scheduling in Kombination mit der Datenratenbegrenzung funktionieren könnte, ist die bei der die entsprechende Qdisc zur Datenratenbegrenzung, also tbf, HTB oder HFSC, vor die TAPRIO_MININET Qdisc gesetzt wird. Somit wird eine verschachtelte Qdisc definiert bei der TAPRIO_MININET nicht an Root gesetzt ist. Die Funktionsweise dieser verschachtelten Qdisc ist in Abbildung 5.7 zusehen. Daran kann gesehen werden das die Funktionsweise von TAPRIO_MININET identische ist zu dem Fall wenn TAPRIO_MININET an Root gesetzt ist. Der einzige Unterschied liegt darin das die Aufrufe von *enqueue()* und *dequeue()* nun entsprechen von der Qdisc zur Datenratenbegrenzung kommen.

Qdisc zur Datenratenbegrenzung als Kind-Qdiscs von TAPRIO

Eine weitere mögliche Umsetzung ist, die Kind-Qdiscs von TAPRIO durch die Qdisc zur Datenratenbegrenzung zu ersetzen. Der angepasste Aufbau von TAPRIO_MININET ist in Abbildung 5.8 dargestellt. Um diesen Aufbau zu erzeugen, wird mit dem tc replace Befehl alle pfifo Qdiscs jeweils durch die entsprechende Qdisc zur Begrenzung der Datenrate ersetzt. Dabei ist die Funktionsweise von TAPRIO_MININET die gleiche wie in Abbildung 5.6 dargestellt ist. Doch diese Umsetzung hat den Nachteil, dass mehrere Qdiscs zur Datenratenbegrenzung benötigt werden. Das dadurch entstehende Problem wird in Abschnitt 7.4.6 genauer erläutert.

Datenratenbegrenzung durch TAPRIO_MININET

Bei dieser Umsetzung wird die Funktionalität zur Datenratenbegrenzung in die TAPRIO_MININET Qdisc integriert. Genauer gesagt wird die *dequeue()* Methode von TAPRIO um die Funktionalität zur Datenratenbegrenzung erweitert. Die Anpassung reguliert dann das nur eine bestimmte Anzahl an Paketen aus TAPRIO entnommen und in die Mininet-Qdisc eingefügt werden können. Durch diese Regulierung ist es dann möglich die Datenrate entsprechend zu begrenzen.

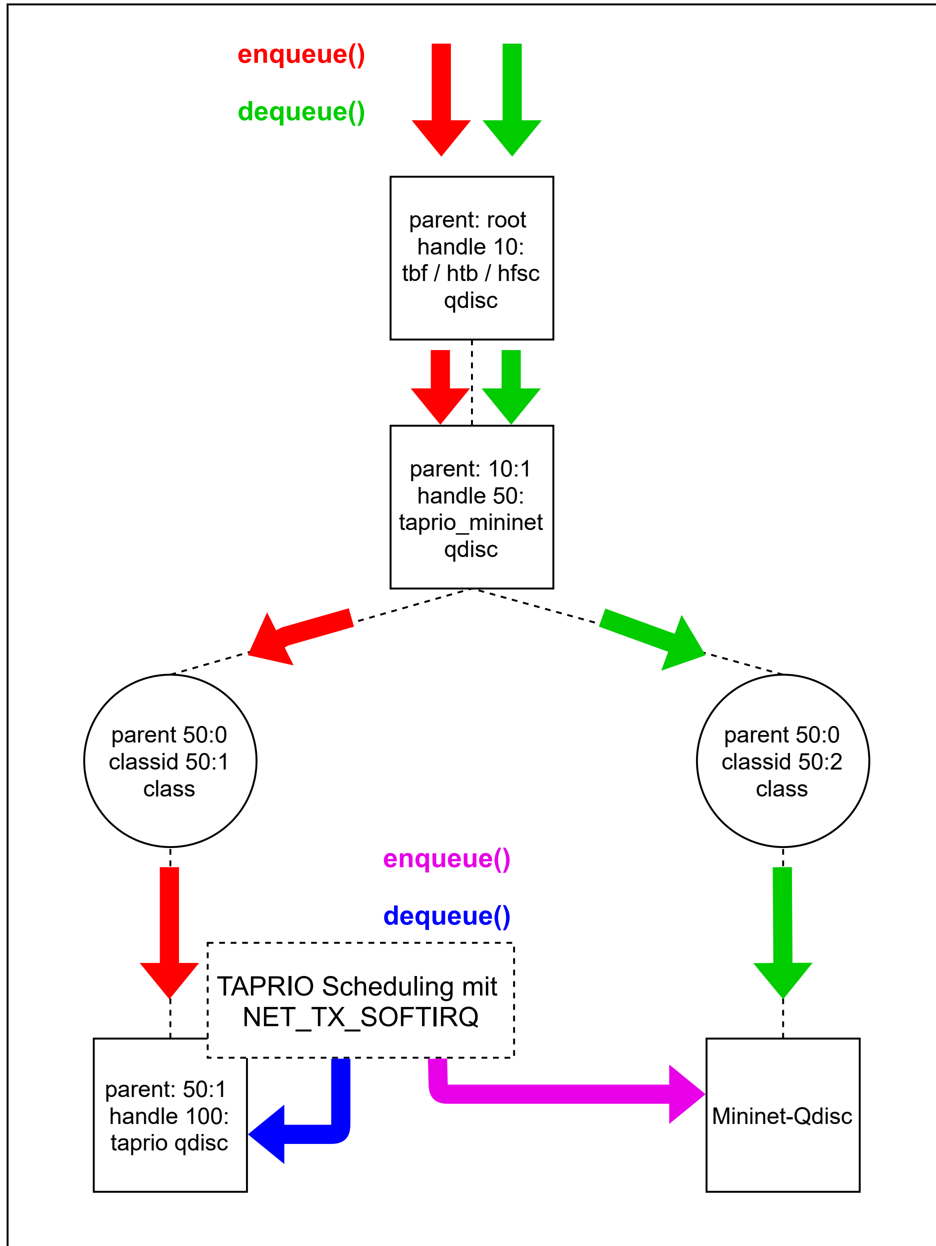


Abbildung 5.7: Funktionsweise TAPRIO_MININET mit Root-Qdisc zur Datenratenbegrenzung

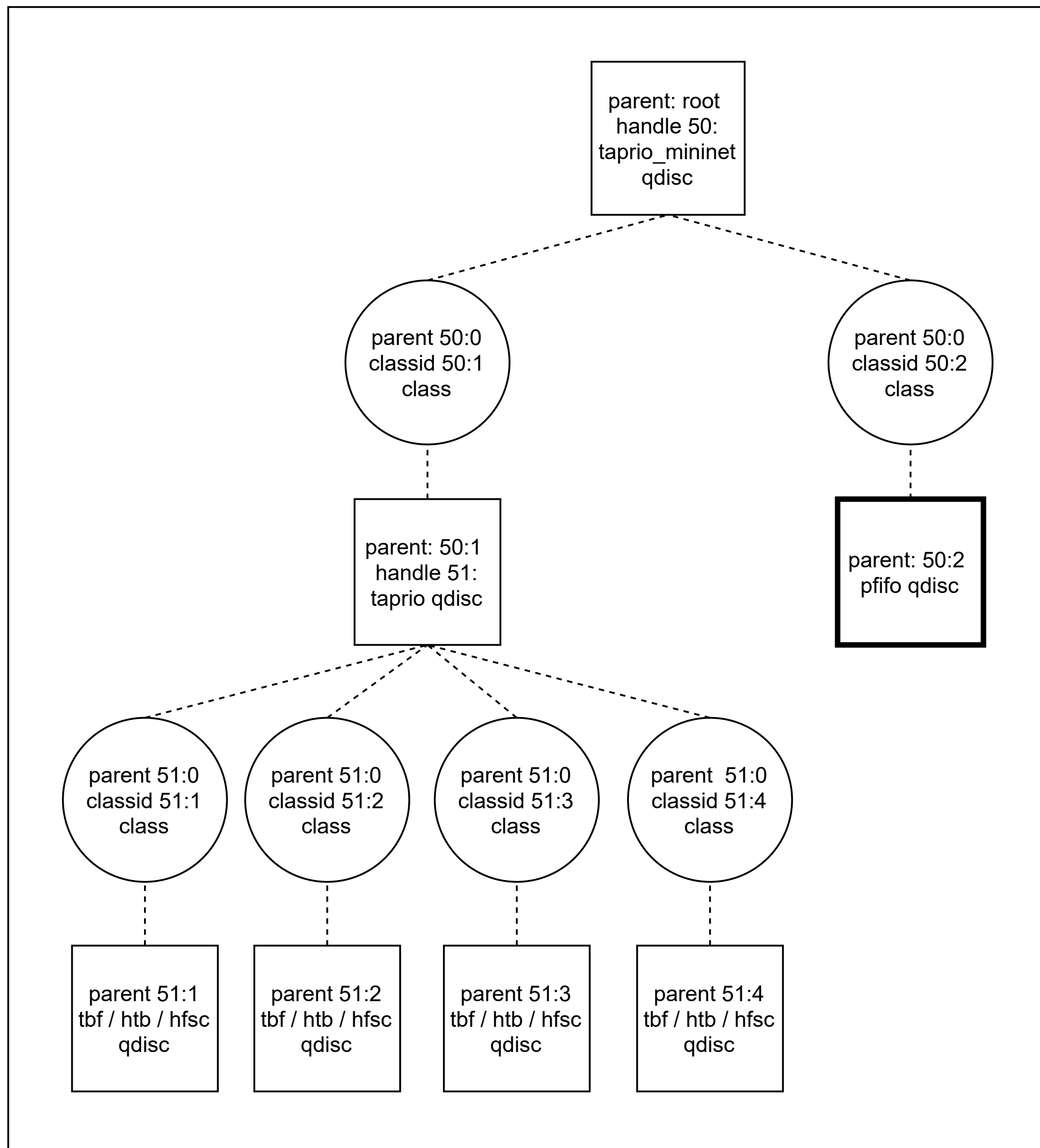


Abbildung 5.8: Aufbau TAPRIO_MININET mit Kind-Qdiscs von TAPRIO zur Datenratenbegrenzung

6 Implementierung

In diesem Kapitel wird die Implementierungen der in Kapitel 5 beschriebenen Konzepte erläutert, damit mit Mininet TSN-Netze emuliert werden können.

6.1 Erweiterung Veth um mehrere TX-Queues

Wie in Abschnitt 5.1 erklärt, lässt sich das veth sehr leicht zu einem Multiqueued-Interface erweitern. Dazu muss nur an der Stelle in Mininet, an der die veth-Paare erzeugt werden, der ip-Befehl entsprechend angepasst werden. Für das Erzeugen der veth-Paare gibt es in Mininet die Methode *makeIntfPair()*, welche in */mininet/util.py* implementiert ist. In dieser Methode muss der ip-Befehl jeweils nach dem Interfacenamen um den Parameter *numtxqueues 8* ergänzt werden. Da sich dieser Befehl in einem if-else-Block befindet, muss die Änderung entsprechend in beiden Blöcken vorgenommen werden.

6.2 Erweiterung Mininet-CLI

Für die Erweiterung des Mininet-CLI um die benötigten Befehle muss die Datei */mininet/cli.py* entsprechend ergänzt werden, wobei für jeden neuen Befehl eine neue Methode definiert werden muss. Dabei muss der Methodennamen folgenderweise aufgebaut sein: *do_<Name>*, wobei der *Name* entsprechend durch den jeweiligen Namen des Befehls ersetzt wird. Mit diesem Namen kann diese Methode aus dem Mininet-CLI aufgerufen werden und die entsprechende Funktionalität wird ausgeführt. Nachfolgend wird die Implementierung der Funktionalität der beiden neuen Befehlstypen, also Filter- und TAPRIO-Befehle, beschrieben. Dabei wird jeweils nur der Fall betrachtet, wenn ein spezielles Interface angegeben wird, denn der Fall mit dem Parameter *all* kann entsprechend aus dem anderen Fall leicht abgeleitet werden.

Damit die benötigten tc- und ip-Befehle im entsprechenden netns, in welchem sich das entsprechende Interface befindet, ausgeführt werden können, gibt es die Methode *cmd(<Befehl>)*. Definiert ist diese in der Klasse Node, welche die Basisklasse für alle Netzwerkelemente, also Host und Switch, darstellt. Diese Klasse hat als Aufgabe hauptsächlich eine Shell im entsprechenden netns zu generieren. Mit dieser Methode wird der angegebene Befehl in der Shell ausgeführt. Der Rückgabewert dieser Methode ist die jeweilige Ausgabe, welche durch die Ausführung des Befehls erzeugt wird.

6.2.1 Filter-Befehle

Die Filter, welche für das Mapping von PCP-Wert auf SKB-Priorität benötigt werden, damit TAPRIO funktioniert, müssen entsprechende gesetzt werden. Um dies für den Benutzer einfach zu machen, werden Befehle zum Hinzufügen und Löschen im Mininet-CLI ergänzt. Außerdem wird noch ein Befehl implementiert, welcher eine Übersicht über die gesetzten Filter ausgibt. Beim *add_filter*-Befehl werden zuerst die Parameter eingelesen. Danach muss der entsprechende Node, in welchem sich das Interface befindet, gefunden werden, damit die benötigten tc-Befehle in der richtigen Shell ausgeführte werden können. Mininet verwaltet dazu eine Datenstruktur, in welcher sich alle Links befinden, wobei die Links die veth-Paare repräsentieren. Um das entsprechende Interface zu finden, wird über diese Datenstruktur iteriert, bis das Interface und somit der Node gefunden wurde. Bevor die Filter hinzugefügt werden können, muss zuerst mit dem tc-Befehl die clsact Qdisc gesetzt werden. Danach können die acht tc-Filter entsprechend gesetzt werden.

Bei der Implementierung des *del_filter*-Befehls liegt der einzige Unterschied in den verwendeten tc-Befehlen, denn hier muss der Befehl zum Löschen der Filter benutzt werden. Um den *show_filter*-Befehl zu realisieren, wird die Klasse Intf, welche die zwei Interfaces eines Links repräsentieren, um die Variable *filter* ergänzt. In dieser wird beim Hinzufügen der Filter entweder Ingress oder Egress gespeichert und beim Löschen wieder entfernt. Um die Übersicht zu generieren, wird über alle Interfaces iteriert. Dies kann gemacht werden, indem über die zwei von Mininet verwalteten Datenstrukturen, welche jeweils alle Hosts und alle Switches beinhalten, iteriert wird. Für jedes dieser Netzwerkelemente wird wiederum über alle Interfaces dieses Elements iteriert, welche sich in einer entsprechenden Datenstruktur befinden. Dabei wird jeder Interfacename gefolgt von der *filter*-Variable ausgegeben.

6.2.2 TAPRIO-Befehle

Damit auch das Hinzufügen und Löschen des TSN-Schedulings durch TAPRIO für den Benutzer einfach ist, werden entsprechende Befehle implementiert. Außerdem wird noch ein Befehl ergänzt, welcher eine Übersicht generiert, an welchen Interfaces TAPRIO gesetzt ist. Beim *add_taprio*-Befehl müssen auch zuerst alle Parameter eingelesen werden. Wie bei den Filtern muss auch zuerst der entsprechende Node gefunden werden. Danach wird geprüft, ob die Mininet-Qdisc an diesem Interface verwendet wird und wenn ja, wird diese gelöscht und TAPRIO_MININET wird gesetzt. Wenn nicht, kann die normale TAPRIO Qdisc verwendet werden. Danach wird noch in dem Fall mit Mininet-Qdisc diese an die vorgesehene Position gesetzt. Dazu muss die Methode *bwCmds()* der TCIntf Klasse angepasst werden, dass es möglich ist, den Parent der Mininet-Qdisc angeben zu können.

Der *del_taprio*-Befehl löscht entsprechend die gesetzt Qdisc, welche das TSN-Scheduling ermöglicht und falls die Mininet-Qdisc verwendet wird, wird diese wieder an Root gesetzt. Um den *show_taprio*-Befehl zu realisieren, wird wieder über alle Interfaces iteriert. Um die benötigte Information zu speichern, wird die Klasse Intf um die Variable *taprio* ergänzt. In diese wird beim Hinzufügen der Name der verwendeten Qdisc zur Realisierung des Schedulings geschrieben und beim Löschen wieder entfernt. Ausgegeben wird wieder jeder Interfacename, gefolgt von der dazugehörigen *taprio*-Variable.

6.3 TAPRIO und Mininet-Qdisc

In diesem Abschnitt wird die Implementierung von zwei Design-Alternativen beschrieben, wie TAPRIO und die Mininet-Qdisc kombiniert werden können. Das wäre einmal die Kombination das TAPRIO unter die Mininet-Qdisc gesetzt wird und zum andern die neue definierte TAPRIO_MININET Qdisc. Außerdem wird noch die Umsetzung beschrieben, wie TAPRIO_MININET erweitert werden muss, sodass diese zusätzlich zum Scheduling auch die Datenratenbegrenzung ermöglicht.

6.3.1 TAPRIO unter Mininet-Qdisc

Damit diese Kombination überhaupt gesetzt und funktionieren kann, muss in der Methode *taprio_init()* der TAPRIO Qdisc zwei Anpassungen vorgenommen werden. Das wäre einmal das TAPRIO nicht nur als Root-Qdisc gesetzt werden kann und zum anderen das der ausgelöste Interrupt durch das periodische Scheduling an der Root-Qdisc greift. Dieser Interrupt wird entsprechend des konfigurierten Schedulings ausgelöst, für Details siehe Abschnitt 5.3.1. Diese beiden Punkte können realisiert werden, indem der if-Block welcher prüft, ob TAPRIO als Root-Qdisc gesetzt wird durch den if-Block in Listing 6.1 ersetzt wird. Dadurch wird ermöglicht das TAPRIO nicht nur als Root-Qdisc gesetzt werden kann, sondern auch unter die Mininet-Qdisc. Falls TAPRIO nicht an Root gesetzt wird, wird in die Variable *q->root* die Root-Qdisc gespeichert, indem diese auf die Qdisc, welche in *dev->qdisc* steht, gesetzt wird. Denn in *dev->qdisc* steht immer die Root-Qdisc des entsprechenden Interfaces. Somit greift der ausgelöste Interrupt des periodischen Schedulings immer an der Root-Qdisc da dieser immer auf der in *q->root* gespeicherten Qdisc greift. Dadurch das der Interrupt an der Root-Qdisc greift, ist der KA, welcher in Abschnitt 5.3.1 definiert ist, wieder geschützt und somit gewährleistet diese Kombination ein funktionierendes Scheduling.

Listing 6.1 Anpassung TAPRIO Qdisc

```
if (sch->parent != TC_H_ROOT)
     q->root = dev->qdisc;
```

6.3.2 TAPRIO_MININET Qdisc-Modul

Bei der Implementierung des TAPRIO_MININET Moduls wird hier hauptsächlich auf zwei wichtige Dinge eingegangen. Das wäre einmal die Umsetzung der TAPRIO *dequeue()* Methode und damit auch, wie Pakete dabei in die Mininet-Qdisc einfügen werden können und zum andern das korrekte Locking, welches benötigt wird, damit die Funktionalität gewährleistet ist.

Bei der Implementierung der *dequeue()* Methode von TAPRIO wird als Grundlage die *__qdisc_run()* Methode verwendet. In der neu implementierten *dequeue()* wird in einer Schleife die ursprüngliche *dequeue()* Methode von TAPRIO ausgeführt und die entnommenen Pakete werden in die Mininet-Qdisc eingefügt. Wie in *__qdisc_run()* gibt es eine Variable *quota*, welche hier mit 64 initialisiert wird. Falls der Wert von *quota* kleiner gleich null wird, weil nach jedem Hinzufügen eines Paketes in die Mininet-Qdisc dieser um eins dekrementiert wird, wird ein Interrupt auf TAPRIO ausgelöst und die Schleife bricht ab. Bei dieser Methode ist es wichtig, dass immer *NULL* zurückgegeben

wird, denn sonst wird das zurückgegebene Paket gesendet. Um die Pakete von TAPRIO aus in die Mininet-Qdisc einfügen zu können, muss auf diese zugegriffen werden können. Dazu wird der *struct taprio_sched* um ein Pointer auf die TAPRIO_MININET Qdisc ergänzt. Mit diesem kann auf die Mininet-Qdisc zugegriffen werden und die Pakete können in diese eingefügt werden. Außerdem muss immer nachdem ein Paket in die Mininet-Qdisc eingefügt wurde, ein Interrupt auf der Root-Qdisc ausgelöst werden. Dies wird benötigt, damit alle Pakete gesendet werden.

Damit die TAPRIO_MININET Qdisc korrekt funktioniert, muss durch ein Locking sichergestellt werden, dass *enqueue()* und *dequeue()* einer Qdisc nicht parallel ausgeführt werden kann. Somit hat TAPRIO_MININET zwei KA. Das ist einmal die Mininet-Qdisc und einmal TAPRIO. Deshalb muss das *enqueue()* in die Mininet-Qdisc mit einem Lock auf dieser Qdisc geschützt werden. Dementsprechend muss dieser Lock auch beim Aufruf von *dequeue()* der Mininet-Qdisc benutzt werden. Wenn *dequeue()* von TAPRIO durch den Interrupt aufgerufen wird, wird das Locking von TAPRIO schon durch den Linux-Network-Stack erledigt. Doch da TAPRIO_MININET mit *enqueue()* Pakete in TAPRIO einfügt, muss dieses Einfügen auch noch mit dem entsprechendem Lock gesichert werden. Somit sind beide KA geschützt und die korrekte Funktionsweise von TAPRIO_MININET ist sichergestellt.

Außerdem ist es noch wichtig, dass in der *enqueue()* Methode von TAPRIO nach dem Einfügen eines Paketes ein Interrupt auf TAPRIO ausgelöst wird. Dadurch wird dann *dequeue()* aufgerufen und Pakete können in die Mininet-Qdisc eingefügt werden. Weiter muss noch das tc-Tool von iproute2 erweitert werden, dass damit die neue TAPRIO_MININET Qdisc überhaupt gesetzt werden kann. Dazu wird die Datei */tc/q_taprio.c* kopiert und zu */tc/q_taprio_mininet.c* umbenannt. In dieser Datei muss dann nur noch der *struct qdisc_util* und die darin enthaltene *id* entsprechend umbenannt werden. Dies ist notwendig, damit das tc-Tool die TAPRIO_MININET Qdisc kennt und die entsprechende Netlink-Nachricht generieren kann.

6.3.3 Integration Datenratenbegrenzung in TAPRIO_MININET

Um die Datenratenbegrenzung mit TAPRIO_MININET zu ermöglichen, muss die ursprüngliche *dequeue()* Methode von TAPRIO angepasst werden. Für diese Anpassung wird als Grundlage die Implementierung der tbf Qdisc verwendet. Dabei wird die Konfiguration der tbf Qdisc so implementiert, wie diese auch von Mininet eingestellt werden kann. Da mit dieser Implementierung nur evaluiert werden soll, ob damit das Scheduling in Kombination mit der Datenratenbegrenzug überhaupt generell ermöglicht ist, wurde versucht, so wenige Anpassungen wie möglich vorzunehmen. Das bedeutet hauptsächlich, dass mit der Implementierung die Datenrate nicht beliebig eingestellt werden kann. Sondern die Parameter von tbf werden hart gesetzt, sodass die Datenrate auf 100 MBit/s begrenzt wird und die Latency auf 5 ms eingestellt ist. Dafür wurde der *struct taprio_mininet_sched* um den Inhalt von *struct tbf_sched_data* erweitert. Um diese Variablen mit den entsprechenden Werten initialisieren zu können, wurden diese Werte aus der tbf Qdisc ausgelesen.

Im Grunde muss einfach die Implementierung der *dequeue()* Methode der tbf Qdisc in die *dequeue()* Methode von TAPRIO kopiert werden. Da aber Mininet die Konfiguration eines *peaks* für die tbf Qdisc nicht unterstützt, kann deswegen der entsprechende if-Block weggelassen werden. Mit dieser Implementierung wird in der *dequeue()* Methode von TAPRIO der Aufruf der *dequeue()* auf den Kind-Qdiscs ersetzt. Entsprechend muss noch implementiert werden, wenn die Datenratenbegrenzung gerade kein Senden erlaubt, das kein Paket entnommen wird und *NULL*

zurückgegeben wird. Die Implementierung der tbf Qdisc verwendet eine Watchdog, welcher aber durch ein Timer ersetzt werden muss, da die Implementierung des Watchdogs mit der Funktionsweise von TAPRIO_MININET nicht kompatibel ist. Dazu wird eine neue Methode definiert, welche ausgeführt wird, wenn die eingestellte Zeit des Timers abgelaufen ist. Diese Methode löst dann ein *NET_TX_SOFTIRQ* Interrupt auf TAPRIO aus. Somit wird die Funktionalität des Watchdogs mithilfe eines Timers realisiert.

7 Evaluation

In diesem Kapitel wird die Erweiterung von Mininet evaluiert und ob die vorgeschlagenen Konzepte die geforderten Funktionalitäten ermöglichen.

7.1 Durchführung der Messungen

Alle Messungen wurden auf einem PC mit einer CPU des Typs *Intel(R) Xeon(R) CPU E5-1650 v3 @ 3.50GHz* durchgeführt. Diese hat sechs physische Kerne und durch Hyper-Threading wird insgesamt auf 12 Kerne kommen. Für die Messungen wurde ein Client verwendet, welcher über ein Socket UDP-Pakete sendet. Dieser sendet immer abwechselnd Pakete mit PCP-Wert eins und null. Um den Sendezeitpunkt der Pakete zu bekommen, wird vor dem Senden die aktuelle Zeit in den Payload geschrieben. Außerdem wird noch eine ID eingefügt. Jedes Paket hat somit eine Größe von 70 Bytes. Bei jeder durchgeführten Messung wurden 1.000.000 Pakete gesendet, wobei die ersten 25.000 Pakete wegen einem möglichen Anlaufverhalten nicht berücksichtigt werden.

Die Messungen wurden mit unterschiedlichen Setups durchgeführt. Dabei wird im Folgenden zur jeder Messung das verwendete Setup beschrieben. Bei allen verwendeten Setups werden immer die Egress Filter verwendet, weil diese an allen Interfaces das Mapping, welches für die Funktionalität von TAPRIO benötigt wird, umsetzten können. Für eine genauere Erklärung siehe Abschnitt 5.2. Die Ingress Filter können nur an Interfaces, welche zu Switches gehören, verwendet werden. Diese gewährleisten aber auch das Mapping siehe dazu die Evaluation in [3].

7.2 Leistungsfähigkeit Veth

Mininet verwendet für Links die veth-Paare. Die Leistungsfähigkeit dieser virtuellen Verbindungen hat daher unmittelbaren Einfluss auf die Leistungsfähigkeit der gesamten Netzemulation. Daher wird im Folgende analysiert, wie die Übertragungszeit über ein veth-Paar ist. Also wie lange die Pakete brauchen, bis diese auf dem entsprechend anderen Interface des Paares empfangen werden.

Dazu wurde eine Messung mit dem Setup welches in Abbildung 7.1 zusehen ist, durchgeführt. Dabei verbindet ein veth-Paar zwei netns miteinander und der Client wird in *ns0* ausgeführt, welcher die Pakete nach *ns1* sendet. Bei der Auswertung wird die Übertragungszeit als Differenz von Ankunftzseitpunkt und Sendezeitpunkt berechnet.

In Abbildung 7.2 sind die Übertragungszeiten aller Pakete dargestellt, wobei Abbildung 7.2a alle Übertragungszeiten zeigt. In Abbildung 7.2b sind die Ausreißer, also die Übertragungszeiten, welche größer sind als 1.5×IQA (Interquartilsabstand), nicht dargestellt. Wobei der IQA durch die Größe der Box gegeben ist. Der Median ist als orangener Strich dargestellt und das grüne Dreieck

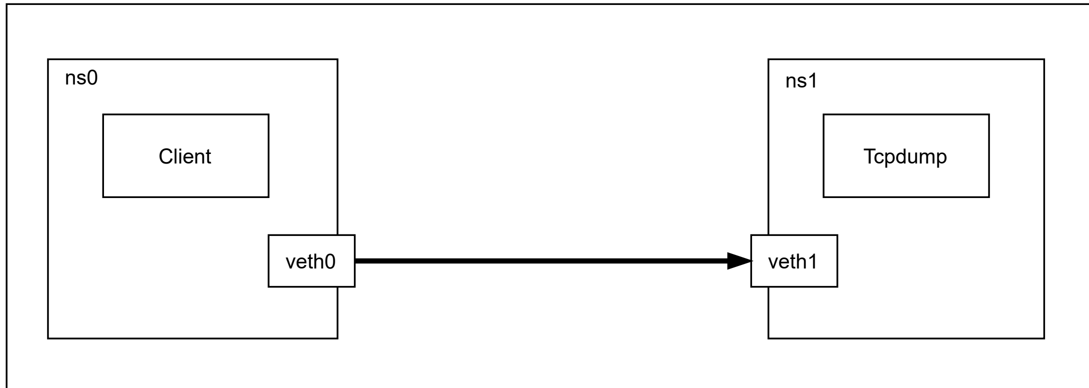
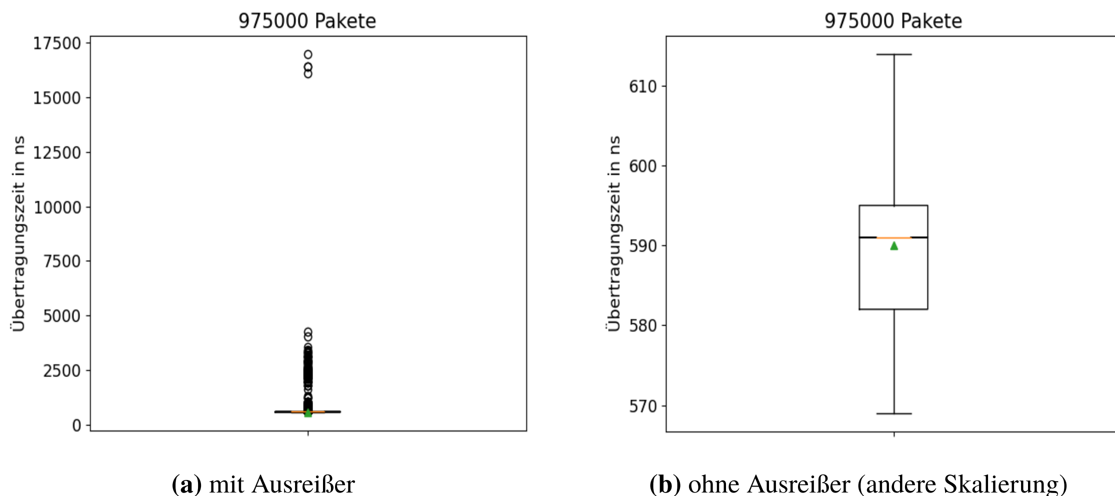


Abbildung 7.1: Setup Veth



(**a**) mit Ausreißer

(**b**) ohne Ausreißer (andere Skalierung)

Abbildung 7.2: Übertragungszeit mit Veth

gibt den Mittelwert an. Außerdem ist noch das 95 % Konfidenzintervall des Medians dargestellt, doch dieses ist so klein, das es nur als schwarzer Strich auf dem Median zu erkennen ist. Der Anteil der Ausreißer beträgt circa 0,31 ‰ von allen Übertragungszeiten. In Abbildung 7.3 ist das Histogramm der Ausreißer dargestellt. Daraus kann gesehen werden, dass die Ausreißer über die gesamte Laufzeit der Messung verteilt sind. Wobei es aber Zeitpunkte gibt, zu denen es sehr wenigen Ausreißer gibt. Eine mögliche Erklärung dafür könnte die Auslastung des PCs sein, auf dem die Messung durchgeführt wurde.

7.3 Mininet ohne Mininet-Qdisc

Als Nächstes wird der Fall betrachtet, wenn Mininet ohne die Emulation von Link-Eigenschaften verwendet wird. Also ohne Mininet-Qdisc welche Datenrate oder Verzögerung der Links emuliert. In diesem Fall wird der normale TAPRIO im weiteren Standalone-TAPRIO genannt für das zeitgesteuerte Scheduling verwendet. Um zu zeigen, das Mininet in Kombination mit dem Standalone-TAPRIO funktioniert, wurde eine Messung mit dem Setup in Abbildung 7.4 durchgeführt. Das Setup

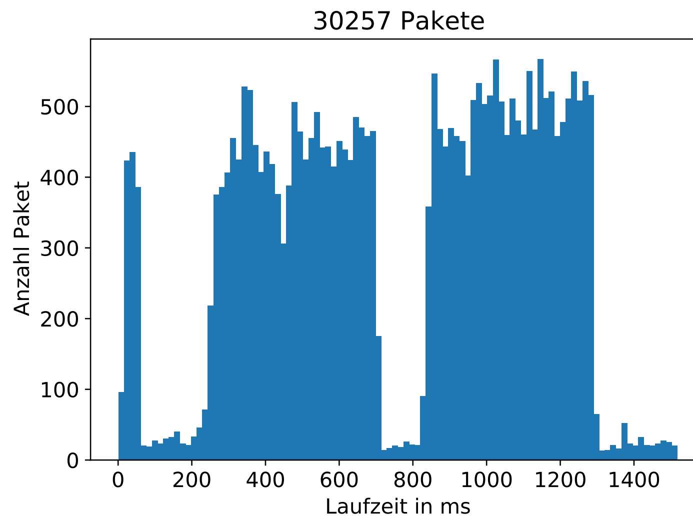


Abbildung 7.3: Histogramm der Ausreißer bei Veth

entspricht der Standard-Topologie von Mininet mit zwei Hosts, welche über einen Switch miteinander verbunden sind. Dabei wird als Switch der OVS verwendet. Wobei die Forwarding-Tabelle mit einem Eintrag gefüllt wird. Dieser Eintrag definiert das alle Pakete, welche über Interface *s1-eth1* empfangen werden, über das Interface *s1-eth2* weitergeleitet werden. Außerdem wird TAPRIO mit einem Zeitfenster von 800 µs für PCP-Wert eins und 200 µs für PCP-Wert null konfiguriert. Dementsprechend beträgt die Zykluszeit 1 ms.

In Abbildung 7.5 ist ein Ausschnitt von drei Zykluszeiten der empfangenen Pakete auf *s1-eth2* dargestellt. Wobei für jedes empfangene Paket eine vertikale Linie in der entsprechenden Farbe gezeichnet wird. Dadurch kann gesehen werden, dass TAPRIO in Kombination mit Mininet funktioniert, denn Pakete mit PCP-Wert null kommen nur in dem definierten Zeitfenster von 200 µs an, welcher blau dargestellt ist. Entsprechend kommen Pakete mit PCP-Wert eins nur im Zeitfenster von 800 µs an, welcher rot dargestellt ist. Im roten Zeitfenster können zwei verschiedene Bereiche gesehen werden. Das ist einmal der Anfang, welcher aus einem komplett roten Bereich besteht und der Rest, in welchem zwischen den roten Strichen immer weiße Striche vorhanden sind. Dies spiegelt das korrekte Verhalten wieder, denn wenn das entsprechende Gate aufgeht, werden zuerst alle Pakete gesendet, welche angekommen sind, als das Gate geschlossen war. Dadurch kommt es zu Beginn des Zeitfensters zu dem komplett roten Bereich, welcher zeigt, dass in diesem viele Pakete empfangen wurden. Die weißen Streifen zu Beginn des blauen Zeitfensters kommen wahrscheinlich durch Ungenauigkeiten zustande. In Abbildung 7.5 kann außerdem der definierte Zyklus des Schedulings von 1 ms gesehen werden.

7.4 TAPRIO_MININET Qdisc

In diesem Abschnitt wird die Funktionalität der neu definierten TAPRIO_MININET Qdisc untersucht. Dazu wurde jede Funktionalität erst einzeln getestet, indem mit dem Setup in Abbildung 7.6 eine Messung durchgeführt wurde. Das Setup besteht aus zwei netns, welche mit einem veth-Paar

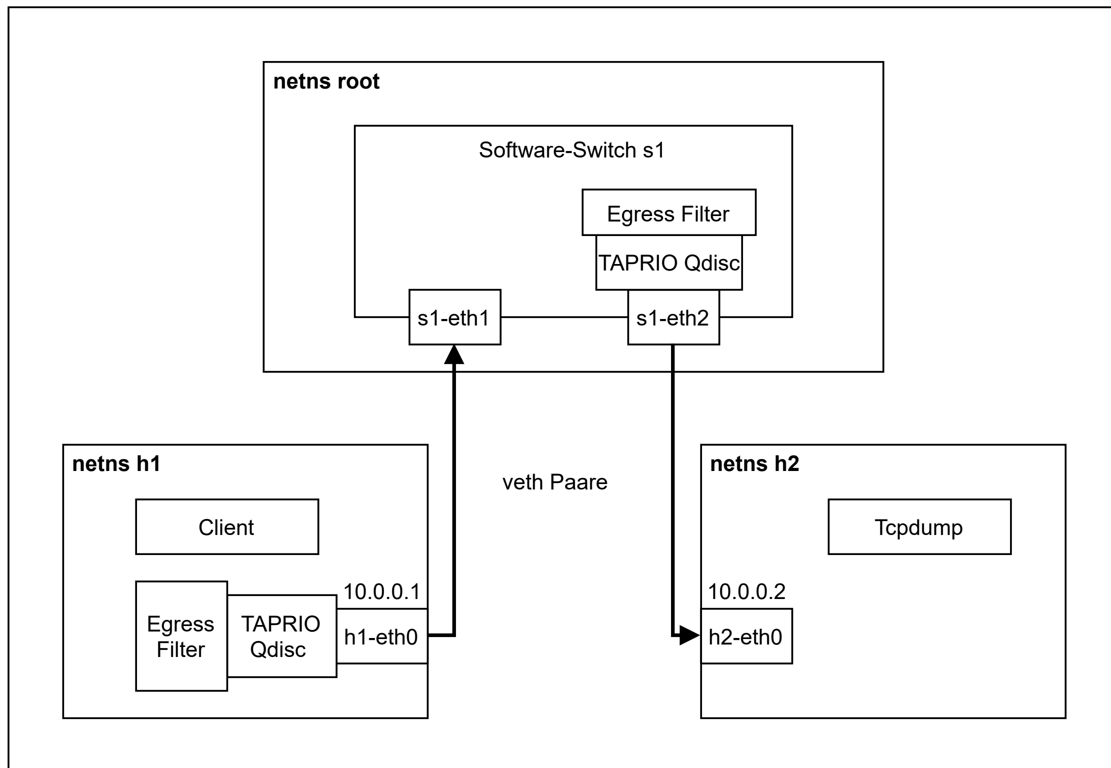


Abbildung 7.4: Mininet Setup

verbunden sind. Im netns, in dem der Client die Pakete sendet, wird am Interface TAPRIO_MININET und die Egress Filter gesetzt. Das Setup wurde gewählt, da dieses grundlegend dem Aufbau von Mininet entspricht, also aus netns und veth besteht. Wenn die Mininet-Qdisc getestet wurde, wurde diese an die entsprechende Stelle von TAPRIO_MININET gesetzt. Anschließend wird noch die Kombination dieser Funktionalitäten getestet.

7.4.1 Verzögerung

Für die Bestimmung der Verzögerung, welche durch TAPRIO_MININET entsteht, wurde eine Messung durchgeführt, bei der TAPRIO_MININET so konfiguriert wurde, dass alle Gates immer offen sind. Somit wird kein TSN-Scheduling umgesetzt. Um die Ergebnisse bewerten zu können, wurde zum Vergleich die gleiche Messung, also mit dem gleichen Scheduling, noch mal mit der Standalone-TAPRIO Qdisc durchgeführt. Dabei wurde in beiden Fällen das Setup in Abbildung 7.6 verwendet, wobei bei der einen Messung TAPRIO_MININET durch die Standalone-TAPRIO Qdisc ersetzt wurde. Bei der Auswertung wird die Verzögerung als Differenz von Ankunftzseitpunkt und Sendezeitpunkt berechnet. Der Grund, warum die Verzögerung evaluiert wird, ist damit über den Overhead der Implementierung im Vergleich zum Standalone-TAPRIO eine Aussage gemacht werden kann. Denn dieser Verzögerung kommt nicht durch die Emulation der Link-Eigenschaften zustande. Somit sollte diese zusätzliche Verzögerung möglichste klein sein.

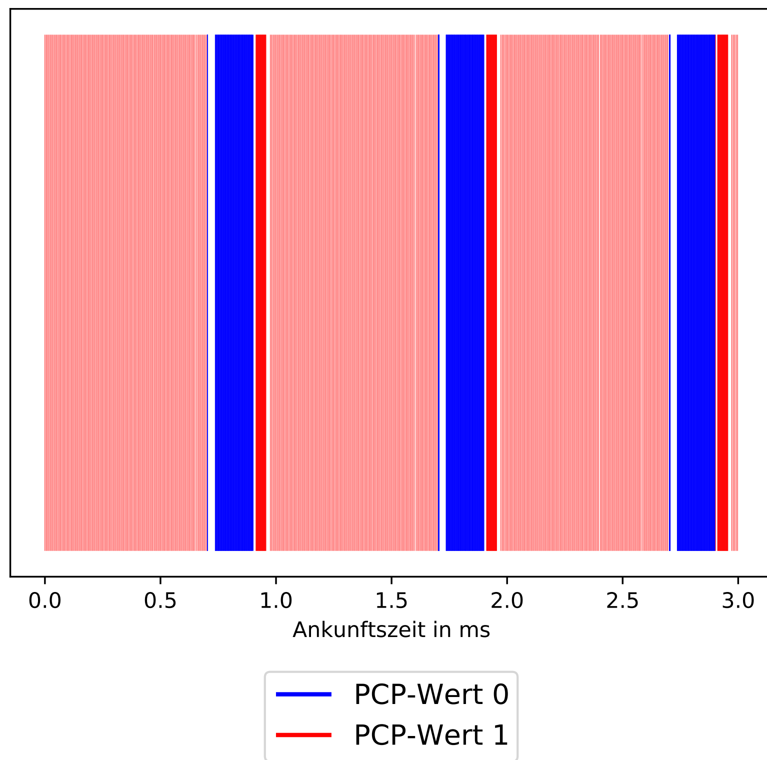


Abbildung 7.5: Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket mit Mininet Setup

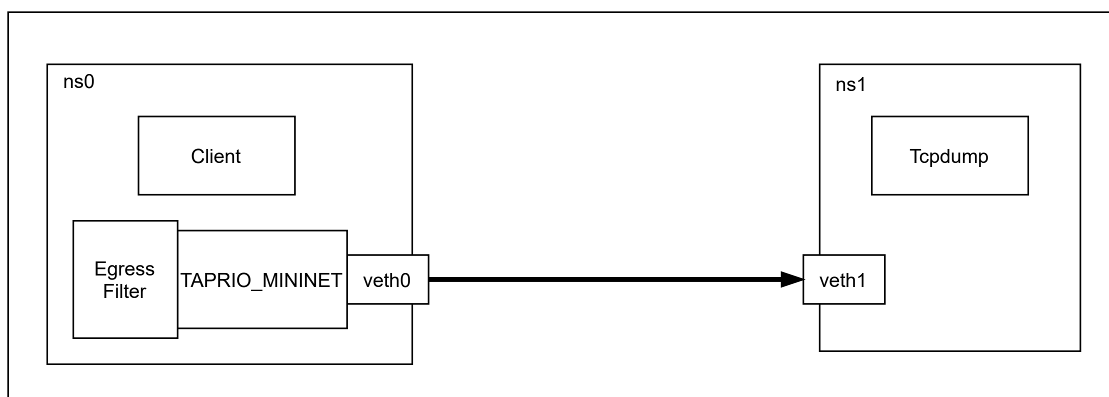
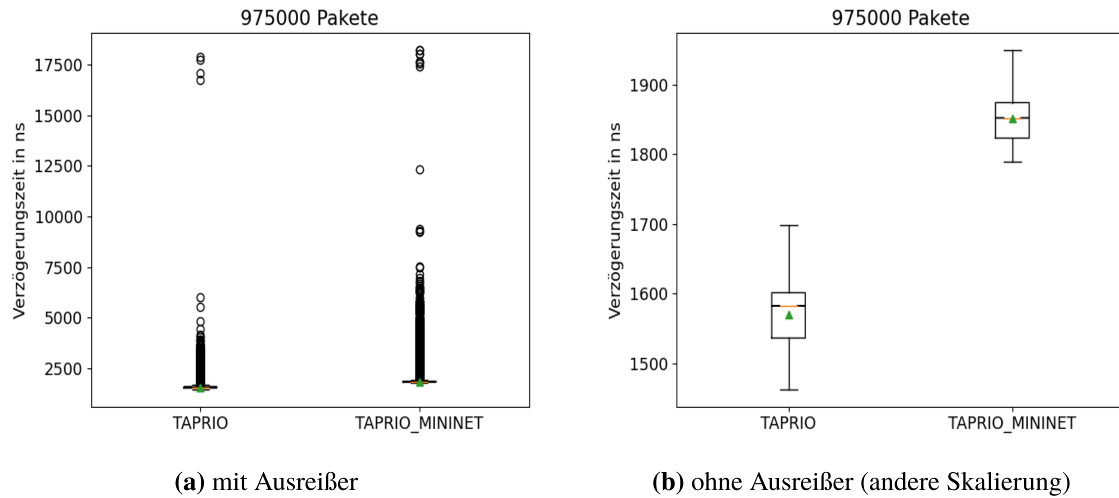


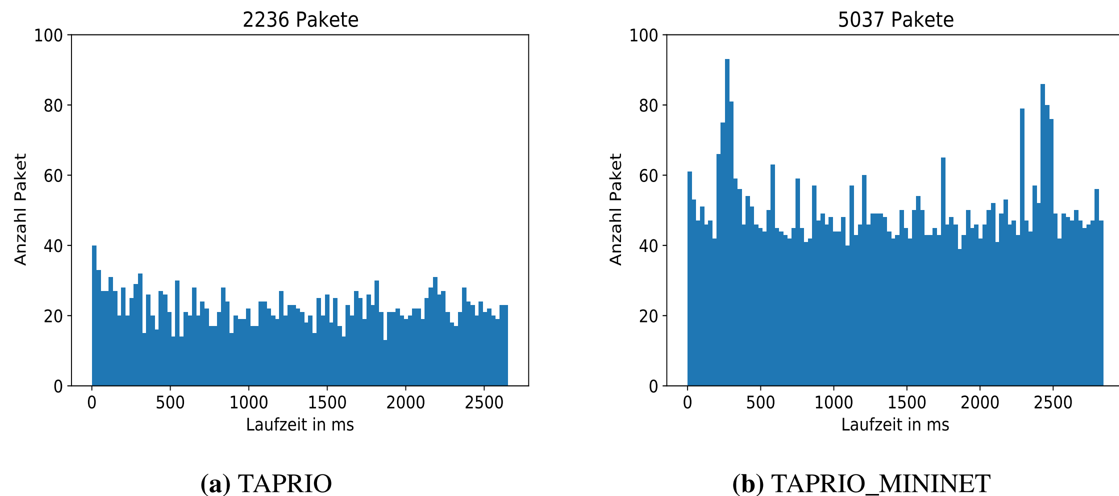
Abbildung 7.6: Grundlegendes Setup für TAPRIO_MININET



(a) mit Ausreißer

(b) ohne Ausreißer (andere Skalierung)

Abbildung 7.7: Vergleich der Verzögerungszeit von TAPRIO mit TAPRIO_MININET



(a) TAPRIO

(b) TAPRIO_MININET

Abbildung 7.8: Histogramm der Ausreißer beim Vergleich der Verzögerungszeit

In Abbildung 7.7a sind die Verzögerungen aller Pakete dargestellt und in Abbildung 7.7b sind wieder die Ausreißer, welche größer als 1.5×IQA nicht dargestellt. Der Mittelwert ist wieder durch ein grünes Dreieck gegeben. Die Sendezeit von TAPRIO_MININET ist demnach in diesem Setup durchschnittlich um circa 281 ns größer als bei der Standalone-TAPRIO Qdisc. Der Anteil der Ausreißer zu allen gemessenen Verzögerungszeiten beträgt bei der Standalone-TAPRIO Qdisc circa 0,023 ‰ und bei TAPRIO_MININET circa 0,052 ‰. Somit ist der Anteil der Ausreißer bei der neue implementierten Qdisc größer geworden im Vergleich zur Standalone-TAPRIO Qdisc. In Abbildung 7.8 ist die Verteilung dieser Ausreißer von beiden Messungen über die Laufzeit der Messung dargestellt. Daraus kann gesehen werden, dass diese bei beiden Messungen über die gesamte Laufzeit verteilt sind. Somit entstehen die Ausreißer nicht durch ein Einschwingverhalten, also zum Beispiel wenn die Caches zu Beginn leer sind.

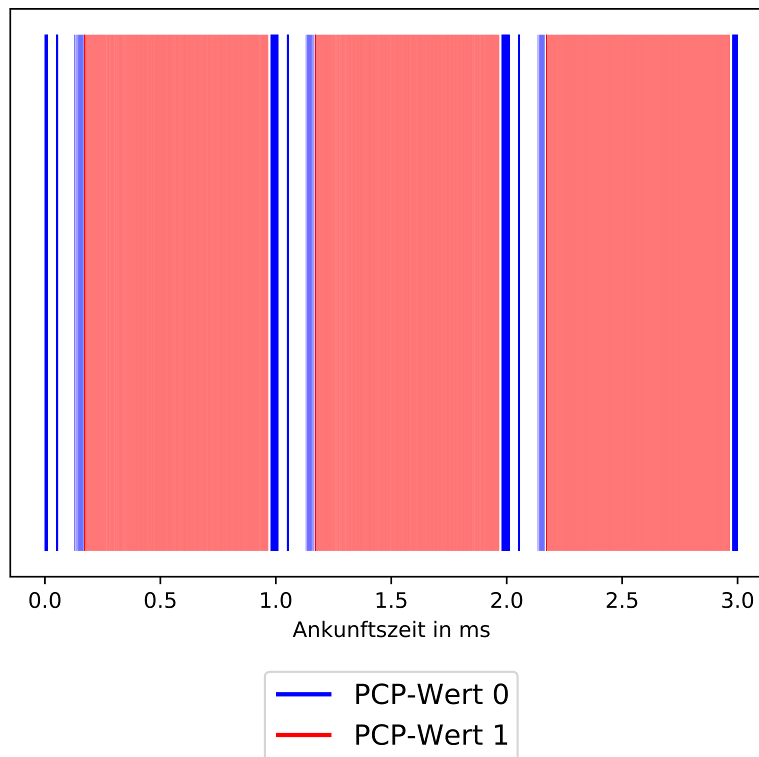


Abbildung 7.9: Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket mit TAPRIO_MININET Setup

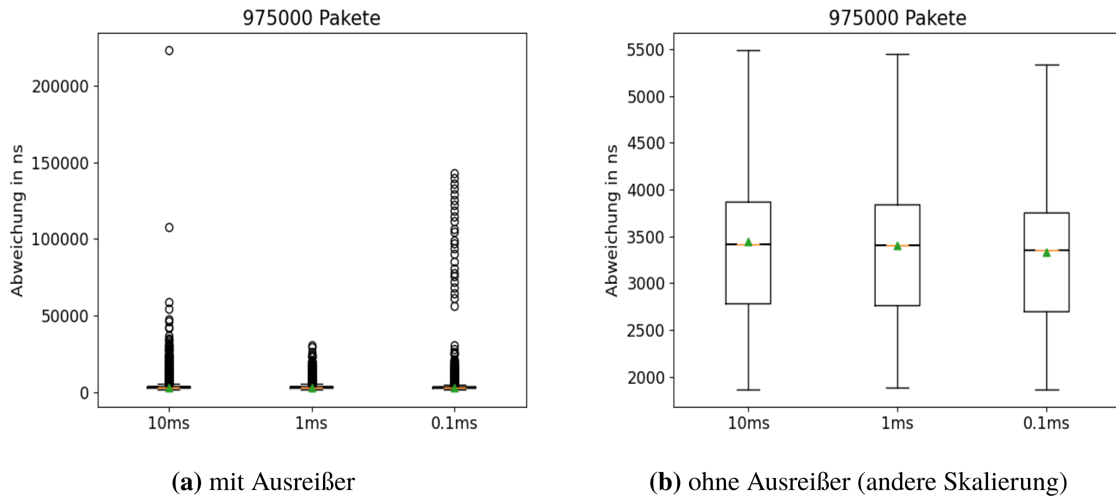
7.4.2 TAPRIO Funktionalität

Um zu zeigen, dass die TAPRIO_MININET Qdisc die TAPRIO Funktionalität hat, also TSN-Scheduling ermöglicht, wurde eine entsprechende Messung mit dem Setup in Abbildung 7.6 durchgeführt. Bei dieser wurde keine Mininet-Qdisc gesetzt. Als Scheduling wurde wieder ein Zeitfenster von 800 µs für PCP-Wert eins und 200 µs für PCP-Wert null eingestellt und somit auch wieder eine Zykluszeit von 1 ms.

In Abbildung 7.9 ist ein Ausschnitt von drei Zykluszeiten der empfangenen Pakete dargestellt, wodurch gesehen werden kann, dass die TAPRIO_MININET Qdisc das zeitgesteuerte Scheduling ermöglicht. Denn die Pakete mit PCP-Wert null werden nur in dem definierten Zeitfenster von 200 µs empfangen. Gleiches gilt auch für die Pakete mit PCP-Wert eins, denn auch diese werden nur im definierten Zeitfenster von 800 µs empfangen. Außerdem kann die definierte Zykluszeit von 1 ms gesehen werden, weil sich die Zeitfenster danach entsprechend wiederholen.

7.4.3 Netem Funktionalität

Bei allen bisherigen Messungen war keine Mininet-Qdisc gesetzt. Um zu zeigen, dass die TAPRIO_MININET Qdisc in Kombination mit netem funktioniert, wurden entsprechende Messungen durchgeführt. Dabei wurde nur die Funktionalität der Emulation der Link-Verzögerung



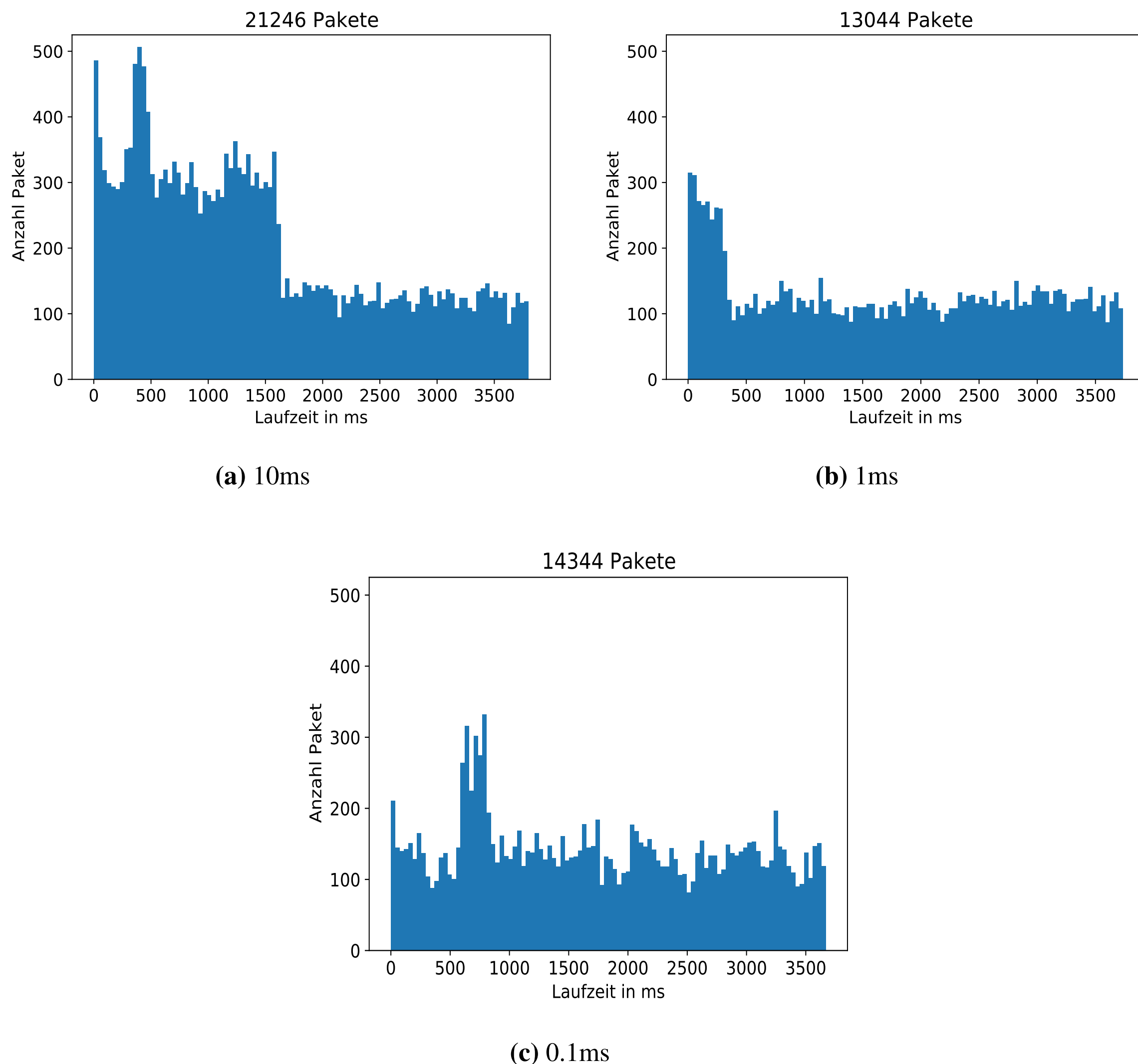
(a) mit Ausreißer

(b) ohne Ausreißer (andere Skalierung)

Abbildung 7.10: Abweichung von der eingestellten Link-Verzögerung mit Netem in Kombination mit TAPRIO_MININET

untersucht, da diese die Wichtigste ist, um reale Netze emulieren zu können. Dazu wurde im Setup Abbildung 7.6 netem an die entsprechende Stelle von TAPRIO_MININET gesetzt. TAPRIO wurde dabei so konfiguriert, dass alle Gates dauerhaft offen sind. Da die Messungen mit 1.000.000 Pakete durchgeführt wurden, wurde das Limit von netem auf diese Anzahl gesetzt. So wurde verhindert, das Pakete durch netem gedroppt wurden und somit immer alle Pakete für die Auswertung verwendet werden können. Es wurden Messungen mit verschiedenen emulierten Link-Verzögerungen durchgeführt.

In Abbildung 7.10 sind die Abweichungen zur eingestellten Link-Verzögerung dargestellt. Wobei Abbildung 7.10a die Abweichungen aller Pakete zeigt und in Abbildung 7.10b sind alle Abweichungen, welche größer als 1.5×IQA sind, nicht dargestellt. Die Abweichung wird berechnet, indem von der gemessenen Verzögerungszeit die eingestellte Link-Verzögerung abgezogen wird. Dadurch kann gezeigt werden, dass TAPRIO_MININET in Kombination mit netem die konfigurierte Link-Verzögerung erzeugt. Die Abweichung gibt an, wie genau die Link-Verzögerungen eingestellt werden können. Bei allen drei Konfigurationen ist die durchschnittliche Abweichung gleich, wie in Abbildung 7.10b an den grünen Dreiecken, welche den Mittelwert angeben, gesehen werden kann. Somit ist die Genauigkeit unabhängig von der eingestellten Link-Verzögerung. Bei den drei konfigurierten Link-Verzögerung liegt der Anteil der Ausreißer zu allen jeweils gemessenen Abweichungen zwischen 0,22 - 0,13 ‰. In Abbildung 7.11 sind die Histogramme der Ausreißer der drei Messungen dargestellt. Daran kann gesehen werden, dass über die gesamte Laufzeit der Messung immer Ausreißer entstehen. Zu Beginn gibt es bei allen drei Messungen mehr Ausreißer. Besonders auffällig sind diese am Anfang der Messung mit eingestellter Link-Verzögerung von 10 ms. Dies kann auf ein Einschwingverhalten zurückgeführt werden, also zum Beispiel das die Caches zu Beginn leer sind.



(a) 10ms

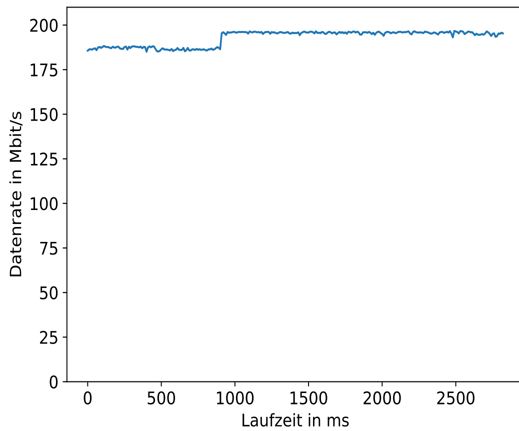
(b) 1ms

(c) 0.1ms

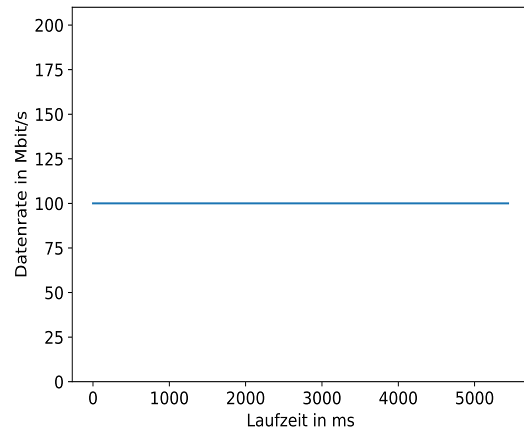
Abbildung 7.11: Histogramme der Ausreißer von TAPRIO_MININET mit Netem

7.4.4 Begrenzung der Datenrate

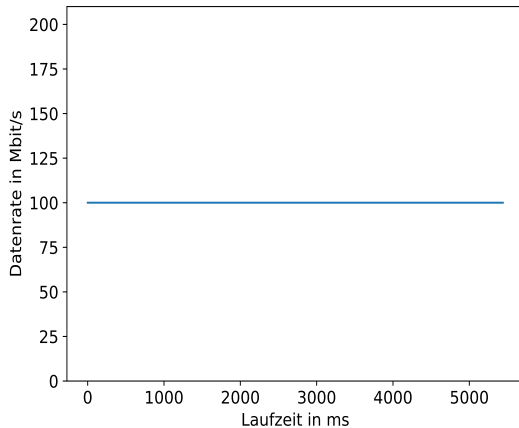
Mit Mininet kann die Datenrate durch drei verschiedenen Qdiscs nämlich tbf, HTB oder HFSC begrenzt werden. Um zu zeigen, dass diese drei in Kombination mit TAPRIO_MININET funktionieren, wurde mit jeder Qdisc eine Messung durchgeführt. Dabei wurde jeweils die Begrenzung der Datenrate auf 100 MBit/s eingestellt. Außerdem wurde bei der tbf Qdisc noch die Latency auf 5 ms eingestellt, da dies auch in Mininet gemacht wird. Damit gezeigt werden kann, dass die Datenrate tatsächlich begrenzt wurde, wurde zuerst eine Vergleichsmessung ohne eine Begrenzung der Datenrate durchgeführt. Mit dieser konnte dann die Datenrate bestimmt werden, mit der der Client sendet. Ausgewertet wird, indem die Laufzeit der Messung in Intervalle von 10 ms aufgeteilt wird. In diesen wird die Anzahl empfangener Pakete gezählt. Daraus kann mit der Paketgröße von 70 Bytes die Datenrate in diesen Intervallen ermittelt werden.



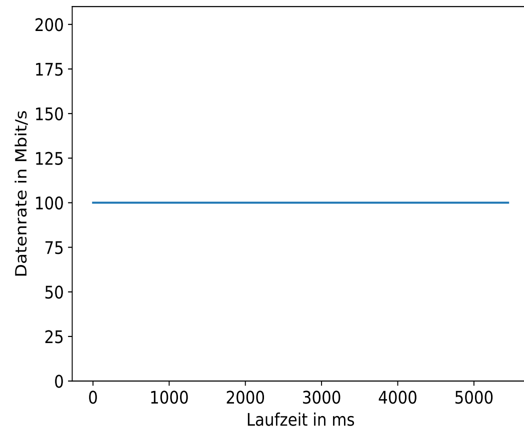
(a) Ohne Begrenzung der Datenrate



(b) mit Tbf



(c) mit HTB



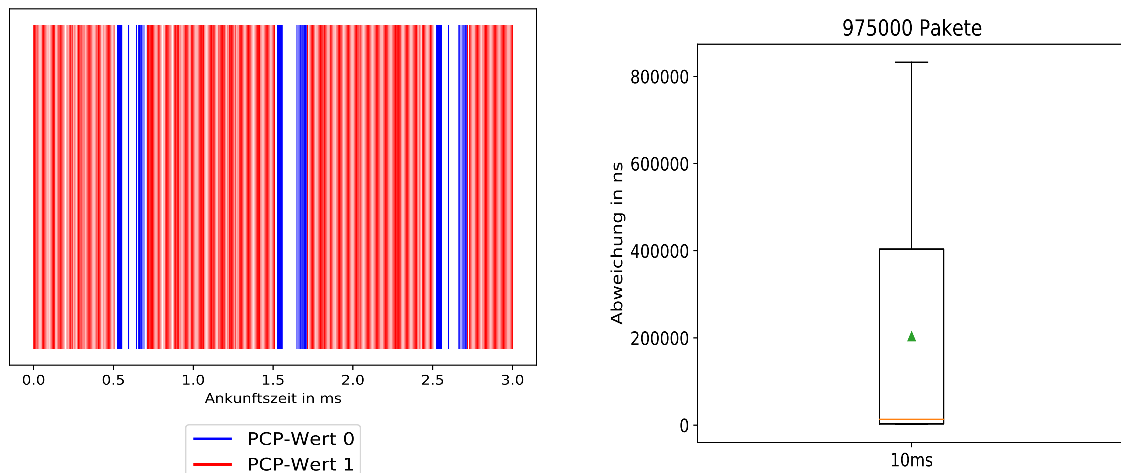
(d) mit HFSC

Abbildung 7.12: Begrenzung der Datenrate in Kombination mit TAPRIO_MININET auf 100 MBit/s

In Abbildung 7.12 ist die errechnete Datenrate über der Laufzeit der verschiedenen Messungen aufgetragen. In Abbildung 7.12a kann gesehen werden, dass der Client mit einer Datenrate von ungefähr 200 MBit/s sendet. Die anderen Abbildungen zeigen jeweils die errechnete Datenrate mit der entsprechend verwendeten Qdisc. Da die gemessene Datenrate der eingestellten Datenrate von 100 MBit/s entspricht, ermöglichen alle drei Qdiscs in Kombination mit TAPRIO_MININET die Begrenzung der Datenrate.

7.4.5 Kombination TSN-Scheduling und Emulation der Link-Verzögerung

Bisher wurde gezeigt, dass mit der TAPRIO_MININET Qdisc die Link-Verzögerung ohne Scheduling emuliert werden kann. Da aber für die Emulation von TSN-Netzen das Scheduling benötigt wird, muss dieses noch in Kombination mit der Link-Verzögerung evaluiert werden. Dazu wurde noch



(a) Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket

(b) Abweichung von der definierten Link-Verzögerung

Abbildung 7.13: Ergebnisse TAPRIO_MININET mit Scheduling und Verzögerung

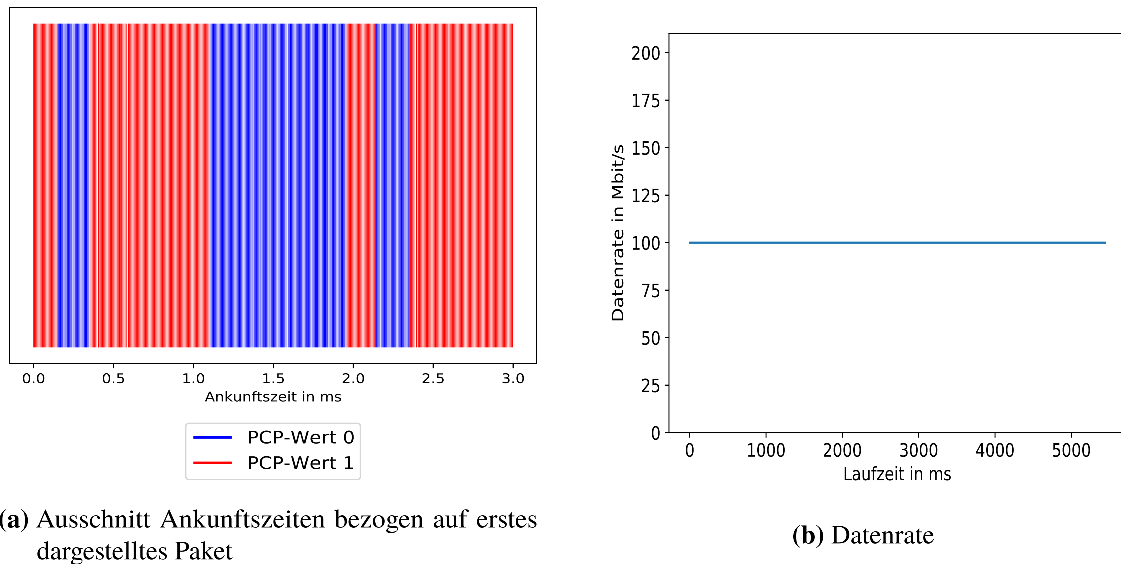
eine Messung mit dem Setup in Abbildung 7.4 durchgeführt, welches aus zwei netns besteht, welche durch ein veth-Paar verbunden sind. Im netns, in welchem der Client läuft, wird TAPRIO mit einem Zeitfenster von 800 µs für PCP-Wert eins und 200 µs für PCP-Wert null konfiguriert. Außerdem wurde mit netem eine Link-Verzögerung von 10 ms eingestellt. Um zu gewährleisten, das keine Pakete verloren gehen, wird noch die maximale Anzahl von Paketen, welche sich in netem gleichzeitig befinden können, auf 1.000.000 gesetzt.

In Abbildung 7.13a ist ein Ausschnitt der empfangenen Pakete dargestellt. An diesem kann gesehen werden, dass das Scheduling korrekt funktioniert. Denn Pakete mit PCP-Wert null kommen nur in dem definierten Zeitfenster von 200 µs an und Pakete mit PCP-Wert eins nur im Zeitfenster von 800 µs. Außerdem ist wird der Scheduling-Zyklus korrekt abgebildet.

In Abbildung 7.13b ist die Abweichung von der definierten Link-Verzögerung dargestellt. Die darin gesehene größte Abweichung spiegelt das eingestellte Scheduling wieder, denn die Pakete können nur gesendet werden, wenn das entsprechende Gate offen ist. Dies führt dazu, dass die Sendezeit dieser Pakete um die entsprechend Zeit, die das Gate zu ist, größer ist. Dabei entsteht die größte Abweichung, wenn ein Paket, kurz nachdem das Gate geschlossen wurde, angekommen ist. Denn dann muss dieses Paket solange warten, bis das Gate wieder aufgeht und dies hängt von dem definierten Scheduling ab. Somit entsteht die größte Abweichung von 800 µs, weil diese die größte Zeit ist, die im Scheduling konfiguriert wurde.

7.4.6 Kombination TSN-Scheduling und Emulation der Datenratenbegrenzung

In diesem Abschnitt wird die Kombination von TSN-Scheduling in Kombination mit der Datenratenbegrenzung evaluiert. Dabei werden die in Abschnitt 5.3.4 beschriebenen Design-Alternativen analysiert, ob mit einer von diesen die Datenratenbegrenzung in Kombination mit dem korrekten Scheduling erreicht werden kann.



(a) Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket

(b) Datenrate

Abbildung 7.14: Ergebnisse Qdisc zur Datenratenbegrenzung in Mininet-Qdisc

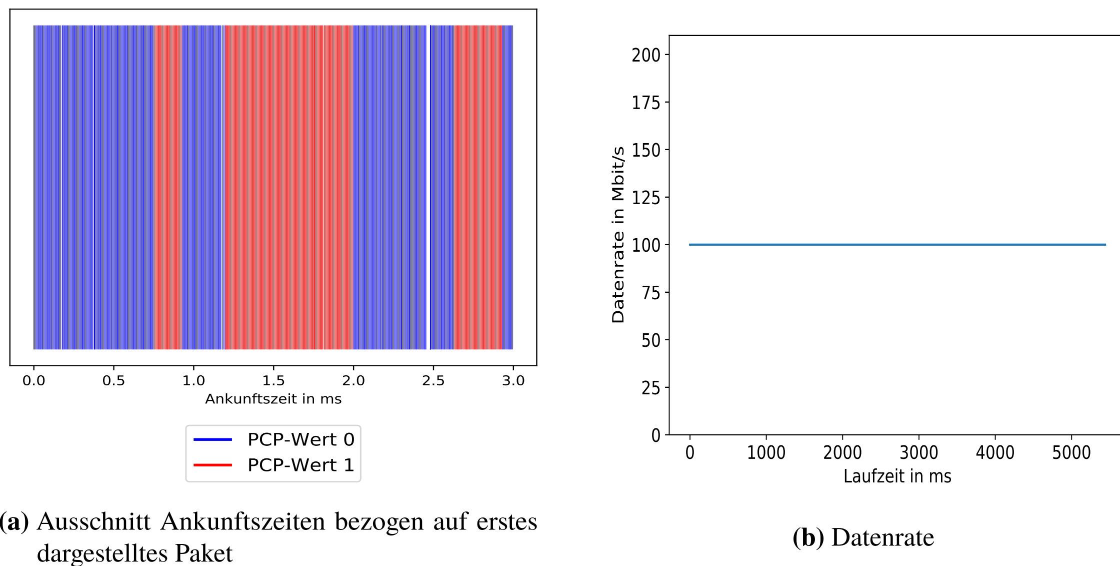
Alle Messungen wurden mit dem Setup, welches in Abbildung 7.6 zusehen ist, durchgeführt. In diesem wurde jeweils die betrachtete Kombination in dem netns mit dem Client an das Interface gesetzt. Bei allen Messungen wurde TAPRIO mit einem Zeitfenster von 800 µs für PCP-Wert eins und 200 µs für PCP-Wert null konfiguriert. Zur Datenratenbegrenzung wurde beispielhaft die tbf Qdisc verwendet, wobei die Begrenzung auf 100 MBit/s eingestellt wurde. Außerdem wurde noch die Latency auf 5 ms eingestellt, um zu gewährleisten, dass keine Pakete verloren gehen. Die Datenrate wird bei allen Messungen bestimmt, indem die Laufzeit der Messung in Intervalle von 10 ms aufgeteilt wird. In welchen dann die Datenrate berechnet wird.

Qdisc zur Datenratenbegrenzung in Mininet-Qdisc

Die erste Mögliche Kombination ist die, bei der die Qdisc zur Datenratenbegrenzung ein Teil der Mininet-Qdisc ist, welche an der entsprechende Stelle von TAPRIO_MININET gesetzt wird.

In Abbildung 7.14b ist die gemessene Datenrate über die Laufzeit der Messung dargestellt. Da diese der eingestellten Datenrate entspricht, kann mit TAPRIO_MININET in Kombination mit der tbf Qdisc die Datenrate entsprechend begrenzt werden.

In Abbildung 7.14a ist ein Ausschnitt der empfangenen Pakete dargestellt. Daran kann gesehen werden, weil die blauen Zeitfenster nicht immer 200 µs groß sind, dass TAPRIO in Kombination mit der Datenratenbegrenzung die Intervalle nicht mehr einhält. Dementsprechend ist auch kein Zyklus mehr erkennbar. Der Grund für dieses Verhalten liegt an der Realisierung der Datenratenbegrenzung. Denn um die definierte Datenrate zu erreichen, werden Pakete durch die entsprechende Qdisc verzögert. Dies führt dann dazu, dass die eingestellten Intervalle nicht mehr eingehalten werden können.



(a) Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket

(b) Datenrate

Abbildung 7.15: Ergebnisse Datenratenbegrenzung vor TAPRIO_MININET

Qdisc zur Datenratenbegrenzung vor TAPRIO_MININET

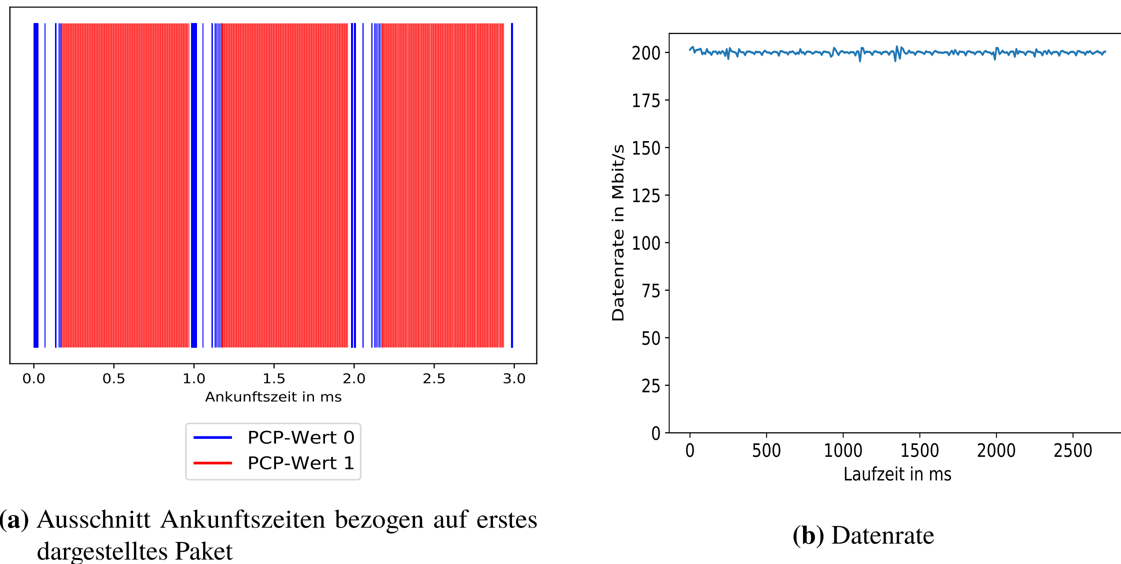
Bei dieser Kombination wird die Qdisc zur Datenratenbegrenzung an Root gesetzt und TAPRIO_MININET wird entsprechend dahinter gesetzt. In Abbildung 7.15b kann gesehen werden, dass die gemessene Datenrate der eingestellten Datenrate entspricht. Somit gewährleistet auch diese Kombination wieder die Begrenzung der Datenrate. Aber auch mit dieser Kombination funktioniert das Scheduling, wie in Abbildung 7.15a gesehen werden kann, nicht mehr korrekt, denn die blauen Zeitfenster sollten immer eine Größe von 200 µs haben. Doch diese sind teils viel größer. Somit werden die definierten Zeitfenster nicht mehr eingehalten.

Qdisc zur Datenratenbegrenzung als Kind-Qdiscs von TAPRIO

Bei dieser Kombination werden die pfifo Qdiscs von TAPRIO_MININET, welche ein Teil von TAPRIO sind, durch tbf Qdiscs ersetzt. Somit werden mehrere tbf Qdiscs zur Datenratenbegrenzung benötigt, welche alle mit 100 MBit/s konfiguriert werden.

In Abbildung 7.16a kann gesehen werden, dass bei dieser Kombination das Scheduling korrekt funktioniert. Denn die definierten Zeitfenster von 200 µs für PCP-Wert null und 800 µs für PCP-Wert eins werden eingehalten. Außerdem kann der definierte Zyklus gesehen werden.

In Abbildung 7.16b ist die gemessene Datenrate dargestellt. Damit kann gesehen werden, dass die eingestellten 100 MBit/s nicht eingehalten werden können. Der Grund dafür ist das mehrere tbf Qdiscs für die Datenratenbegrenzung verwendet werden. Diese führt dazu, dass die Gesamt-Datenrate größer ist als die eingestellte. Bei dieser Messung beträgt die gemessene Datenrate 200 MBit/s obwohl nur eine Datenrate von 100 MBit/s erreicht werden sollte.



(a) Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket

(b) Datenrate

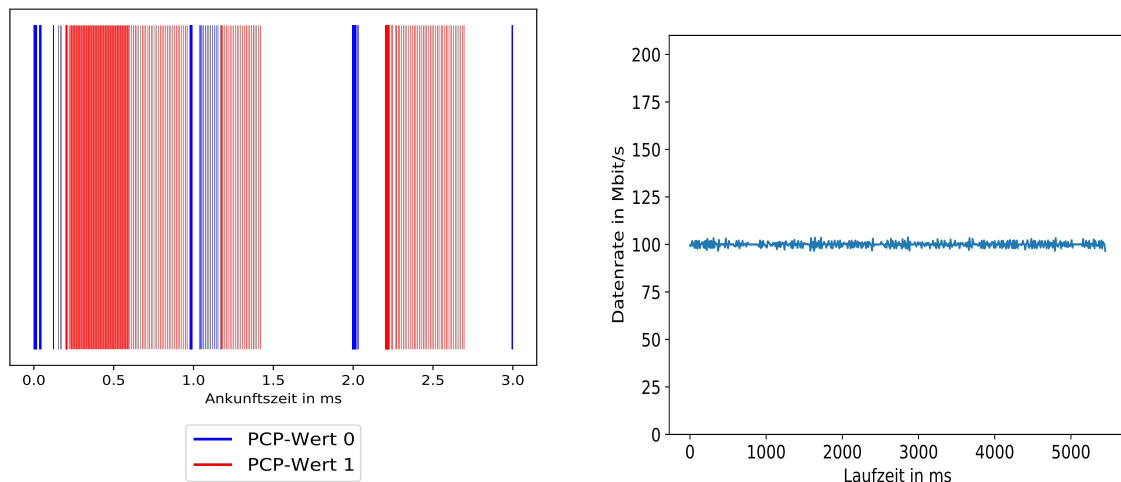
Abbildung 7.16: Ergebnisse Datenratenbegrenzung durch TAPRIO Kind-Qdiscs jeweils auf 100 MBit/s eingestellt

In Abbildung 7.17 sind die Ergebnisse einer weiteren Messung dargestellt. Bei dieser wurden die tbf Qdiscs mit einer Datenrate von jeweils 50 MBit/s konfiguriert. Somit kann erreicht werden, dass die Gesamt-Datenrate auf die geforderten 100 MBit/s wie in Abbildung 7.17b zusehen, begrenzt wurde. Wie in dem Ausschnitt der empfangenen Pakete in Abbildung 7.17a gesehen werden kann, werden dabei die Zeitfenster korrekt eingehalten. Wobei die weißen Streifen, in den keine Pakete empfangen werden, zustande kommen, um die eingestellte Datenrate zu erreichen. Doch diese Umsetzung hat ein Problem. Denn wenn für eine gewisse Zeit nur Pakete einer TC gesendet werden, wird nur die Datenrate erreicht, die in der entsprechenden tbf Qdisc definiert wurde. Dies wäre bei diesem Setup dann 50 MBit/s. Somit wird nur mit der Hälfte der eingestellten Datenrate gesendet. Mit dieser Umsetzung kann dementsprechend keine Lösung erreicht werden, mit welcher die Datenrate in allen Fällen korrekt begrenzt werden kann.

Datenratenbegrenzung durch TAPRIO_MININET

Bei dieser Kombination wird die tbf Funktionalität in TAPRIO_MININET integriert. Die Datenratenbegrenzung wird auf 100 MBit/s eingestellt. Außerdem wird die Latency von tbf auf 5 ms gesetzt.

Wie in Abbildung 7.18b dargestellt ist, kann dadurch die Datenrate auf die 100 MBit/s begrenzt werden. Auffällig ist aber, dass die Datenrate Schwankungen nach oben und unten hat. Der Grund, warum diese entstehen, ist die Funktionsweise der tbf Qdisc. Denn diese ermöglicht es nur, die durchschnittliche Datenrate zu definieren. Dementsprechend gibt es Zeitpunkte, in den die Datenrate größer ist und zu anderen Zeitpunkten kleiner. Dieses Verhalten führt dann in Kombination mit der Funktionsweise des Schedulings zu den Schwankungen. Um diese noch deutlicher zu sehen, wurde die Datenrate noch mal ausgewertet, aber diesmal mit Intervallen der Größe 1 ms. In Abbildung 7.19 ist dieses Auswertung dargestellt und es kann gesehen werden, dass die Schwankungen sehr groß



(a) Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket

(b) Datenrate

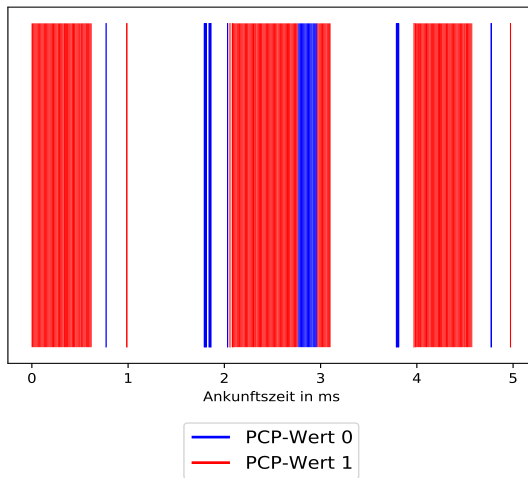
Abbildung 7.17: Ergebnisse Datenratenbegrenzung durch TAPRIO Kind-Qdiscs jeweils auf 50 MBit/s eingestellt

sind. Somit kann mit dieser Kombination die Datenrate nur durchschnittlich begrenzt werden. Wegen den vorhandenen Schwankungen stellt diese Kombination im Bezug auf die Datenratenbegrenzung keine gute Umsetzung dar, denn in realen Links gibt es diese Schwankungen nicht.

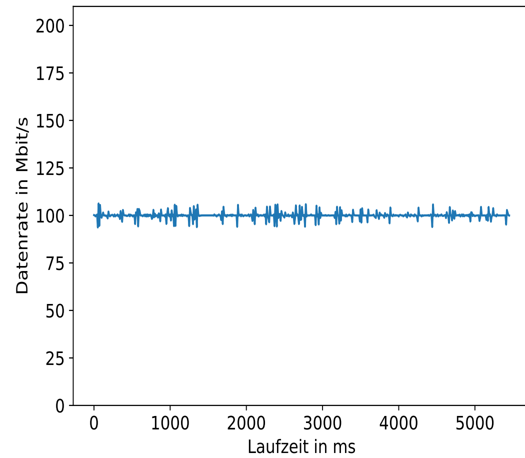
In Abbildung 7.18a ist ein Ausschnitt von fünf Zykluszeiten der empfangenen Pakete dargestellt. Es kann gesehen werden, dass die Intervalle meistens aber nicht immer eingehalten werden. Denn im dargestellten Ausschnitt gibt es eine Stelle, an der das Intervall nicht korrekt eingehalten wird. Dies ist am Zeitpunkt 2 ms, denn hier ist das blaue Intervall ein wenig größer als die eingestellten 200 µs. Der Grund dafür liegt in der Integration der Datenratenbegrenzung in die *dequeue()* Methode von TAPRIO, welche das Senden der Pakete steuert. Da dadurch die Methode um Code ergänzt wurde, welcher zusätzlich ausgeführt werde muss, führt diese dazu, dass die Intervalle nicht immer korrekt eingehalten werden können. Außerdem kann in Abbildung 7.18a das Problem mit der Datenrate gesehen werde. Denn es gibt viele weiße Streifen, in den keine Pakete ankommen, dafür aber Zeitpunkte, zu den sehr viele Pakete empfangen werden. Dies spiegelt die oben erläuterten Schwankungen wieder, welche entstehen, weil die Datenrate nur durchschnittlich begrenzt werden kann. Bei einem realen Link würden die Pakete nämlich kontinuierliche mit konstanter Datenrate über den Link gesendet werden. Wenn diese Verhalten umgesetzt werden könnte, würde dies dazu führen, dass die Pakete im jeweiligen Zeitfenster verteilt empfangen werden. Somit würde es die weißen Streifen im Ausschnitt nicht mehr geben.

7.5 Mininet mit Scheduling und Link-Verzögerung

Um noch abschließend zu zeigen, dass die entwickelte TAPRIO_MININET Qdisc auch in Mininet funktioniert, wurde noch eine Messung mit dem Setup in Abbildung 7.4 durchgeführt. Dabei wurde natürlich TAPRIO durch TAPRIO_MININET ersetzt. Da in diesem Kapitel gezeigt wurde, dass für die Datenratenbegrenzung keine realistische Lösung gefunden wurde, wurde die Messung



(a) Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket



(b) Datenrate

Abbildung 7.18: Ergebnisse Datenratenbegrenzung durch TAPRIO_MININET

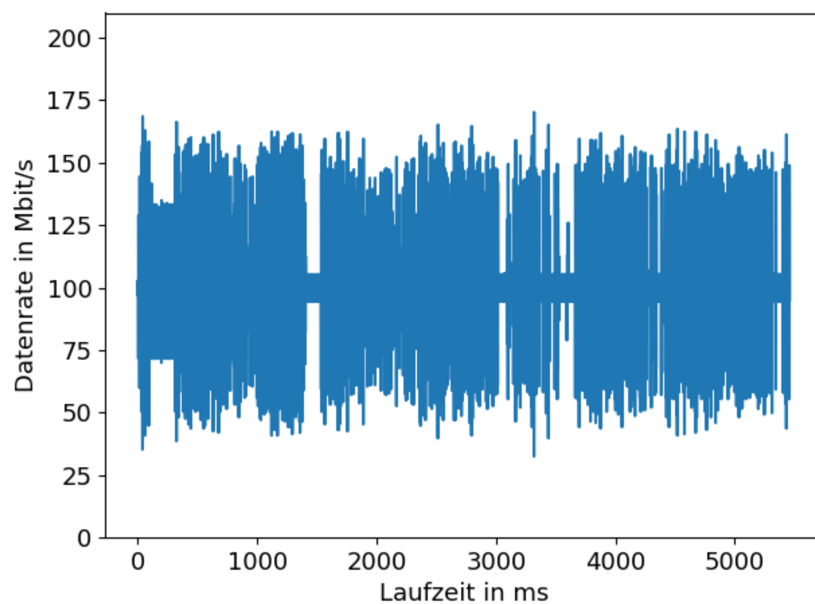
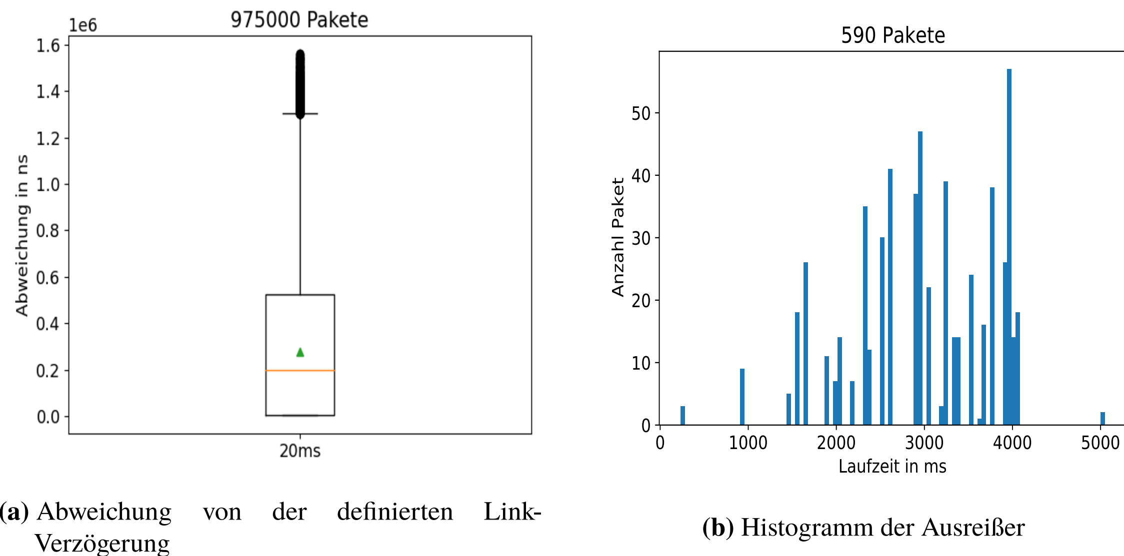


Abbildung 7.19: Datenratenbegrenzung durch TAPRIO_MININET mit Intervallen von 1 ms



(a) Abweichung von der definierten Link-Verzögerung

(b) Histogramm der Ausreißer

Abbildung 7.20: Ergebnisse Datenratenbegrenzung durch TAPRIO_MININET

ohne Begrenzung der Datenrate gemacht. Somit wurde TAPRIO mit einem Zeitfenster von 800 µs für PCP-Wert eins und 200 µs für PCP-Wert null konfiguriert. Außerdem wurde noch eine Link-Verzögerung von 10 ms definiert, wobei aber die Gesamt-Verzögerungszeit 20 ms beträgt, da die Pakete über zwei Links gesendet werden. Die Forwarding-Tabelle des Switches wurde wieder mit dem Eintrag gefüllt, welcher alle Pakete, die über Interface *s1-eth1* empfangen werden, über das Interface *s1-eth2* weiterleitet.

In Abbildung 7.20a ist die Abweichung zur eingestellten Link-Verzögerung von 20 ms dargestellt. An diesen kann wieder das eingestellte Scheduling gesehen werden. Da in diesem Setup die Pakete zweimal durch TAPRIO_MININET müssen, ist die größte Abweichung zweimal das größte eingestellte Zeitfenster des Schedulings. Da das größte konfigurierte Zeitfenster 800 µs groß ist, beträgt die größte Abweichung 1.6 ms wie auch im Diagramm gesehen werden kann. In Abbildung 7.20b sind die Abweichungen, welche größer als 1.5×IQA sind, dargestellt. Diese entsprechend gerade den Paketen, welche jeweils immer ankommen sind, wenn das Gate gerade zu gegangen ist. Somit tritt dieser Fall relativ selten auf, weil es nur wenige Ausreißer gibt.

In Abbildung 7.21 ist ein Ausschnitt der empfangenen Pakete dargestellt. Daran kann gesehen werden, dass die definierten Zeitfenster eingehalten werden. Denn Pakete mit PCP-Wert eins kommen nur im entsprechenden Zeitfenster von 800 µs an. Die weißen Streifen im blauen Zeitfenster sind im Vergleich zum Ausschnitt in Abbildung 7.5 größer. Da der in Abbildung 7.5 dargestellte Ausschnitt die empfangenen Pakete einer Messung mit dem gleichen Setup aber mit gesetztem TAPRIO zeigt, kann daraus geschlossen werden, dass die größeren weißen Streifen durch die Implementierung der TAPRIO_MININET Qdisc in Kombination mit der netem Qdisc für die Link-Verzögerung zustande kommen. Auffallend ist noch das der in Abbildung 7.5 dargestellte Ausschnitt, die gleiche Anzahl an empfangene Paket mit PCP-Wert eins und null hat. Somit kommen in den kleineren blauen Streifen sehr viele Pakete an. Der Grund dafür ist, dass es keine Begrenzung der Datenrate gibt und somit werden die Pakete so schnell gesendet, wie es die Implementierung ermöglicht. Wenn die Datenrate realistisch begrenzt werden könnte, würde dieses Verhalten nicht mehr auftreten. In diesem Fall würden dann die Pakete über das gesamte Zeitfenster verteilt empfangen werden.

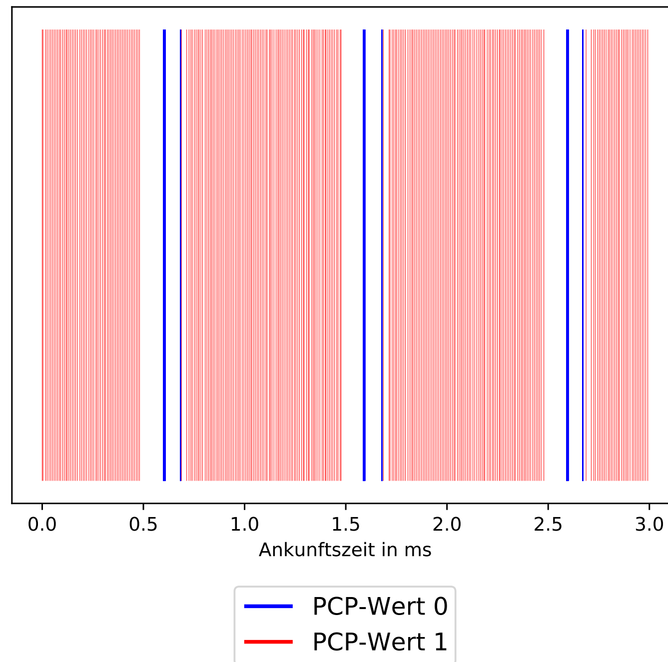


Abbildung 7.21: Ausschnitt Ankunftszeiten bezogen auf erstes dargestelltes Paket

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde Mininet erfolgreich um ein zeitgesteuerten Scheduling-Mechanismus erweitert. Mit dieser Erweiterung können nun TSN-Netze emuliert werden.

Umgesetzt wurde dies, indem die TAPRIO Qdisc in Mininet integriert wurde, welche das zeitgesteuerte Scheduling ermöglicht. Hierzu wurden zunächst die von Mininet zur Link-Emulation verwendeten veth-Paare um mehrere TX-Queues erweitert, sodass TAPRIO diese für das Scheduling verschiedener TCs verwenden kann. Des Weiteren musste ein Mapping von PCP-Wert auf SKB-Priorität umgesetzt werden, denn TAPRIO klassifiziert die Pakete anhand der SKB-Priorität, die Pakete aber die Priorität im PCP-Feld haben. Um dieses Mapping zu realisieren, wurden die in [3] entwickelten tc-Filtern verwendet. Ferner wurde das Mininet-CLI erweitert, sodass mit entsprechenden Befehlen das Setzten und Löschen dieser Filter für den Benutzer so einfach wie möglich ist. Außerdem wurden noch weitere Befehle ergänzt, mit welchen die Konfiguration des Schedulings einfach vorgenommen werden kann.

Schließlich wurden verschiedene Design-Alternativen zur Integration der Qdiscs, zur Emulation der Link-Eigenschaften und der von TAPRIO implementierten Scheduling-Funktion entworfen, implementiert und experimentell verglichen. Dabei lag der Fokus bei der Emulation der Link-Eigenschaften auf der Datenratenbegrenzung und der Möglichkeit, die Link-Verzögerung zu definieren. Denn diese beiden Eigenschaften sind die wichtigsten, damit mit Mininet ein reales Netz emuliert werden kann. Die Schwierigkeit bei dieser Umsetzung war, dass diese der realen Situation entsprechen sollte, also das zuerst das Scheduling erfolgt und danach die Link-Eigenschaften emuliert werden. Als Erstes wurden die zwei nahe liegenden Design-Alternativen betrachtet, bei welchen die Qdiscs jeweils hintereinander geschaltet werden. Doch mit diesen beiden konnte die reale Situation nicht umgesetzt werden. Deshalb wurde die TAPRIO_MININET Qdisc neu entwickelt. Mit dieser kann die reale Situation abgebildet werden. In der Evaluierung konnte gezeigt werden, dass mit der TAPRIO_MININET Qdisc das zeitgesteuerte Scheduling in Kombination mit der Link-Verzögerung emuliert werden kann.

Für die realistische Emulation der beschränkten Link-Datenrate zusammen mit dem zeitgesteuerten Scheduling wurden weitere Design-Alternativen entwickelt. Doch mit diesen konnte keine zufriedenstellende Lösung gefunden werden, die den Link realistisch emuliert. Deshalb ist diese Realisierung ein offenes Problem, das Gegenstand zukünftiger Arbeiten sein könnte. Die beste Umsetzung dieses Problems konnte erreicht werden, indem die Datenratenbegrenzung in die TAPRIO_MININET Qdisc integriert wurde. Dafür wurde die Implementierung der tbf Qdisc verwendet. Mit dieser konnte aber die Datenrate nur durchschnittlich begrenzt werden. Da in Mininet für die Begrenzung der Datenrate auch die HTB und HFSC Qdisc verwendet werden kann, könnte evaluiert werden, ob mit dieser Integration in TAPRIO_MININET eine realistischere Datenratenbegrenzung erreicht werden kann. Falls nicht, müsste eine komplett neue Umsetzung entwickelt werden.

Eine zweite offene Frage ist, inwieweit das emulierte Verhalten dem Verhalten eines realen TSN-Netzes entspricht. Aktuell verlässt sich die Implementierung auf die Software-Implementierung von TAPRIO. Eine quantitative Analyse, welche Präzision und Genauigkeit dieser Software-Implementierung im Vergleich zu einer Hardware-Implementierung erzielt, ist Gegenstand zukünftiger Arbeiten.

Literaturverzeichnis

[1] F. Duerr. *„Software TSN-Switch with Linux“*. Apr. 2019. URL: https://www.frank-durr.de/?p=376 (besucht am 11. 03. 2021) (zitiert auf S. 17, 26).

[2] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, K. Rothermel. „NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++“. In: *2019 International Conference on Networked Systems (NetSys)*. 2019, S. 1–8 (zitiert auf S. 33).

[3] J. Herrmann, M. Hildebrand, M. Knorpp, M. Vukovic. *„Design and Performance Evaluation of a Linux Software-Switch for Time-Sensitive/Software-Defined Networking (TSSDN)“*. Bachelor Research Project. Universität Stuttgart. 2020 (zitiert auf S. 33, 36, 37, 57, 75).

[4] *HFSC Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/tc-hfsc.8 (besucht am 14. 03. 2021) (zitiert auf S. 23).

[5] *HTB Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/tc-htb.8 (besucht am 14. 03. 2021) (zitiert auf S. 23).

[6] *ip Link Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/4712a4617408da5afabc9433c7316a99363fd053/man/man8/ip-link.8.in (besucht am 14. 03. 2021) (zitiert auf S. 27).

[7] *iproute2 v5.11.0 GitHub*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/tree/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef (besucht am 14. 03. 2021) (zitiert auf S. 21).

[8] G. N. Kumar, K. Katsalis, P. Papadimitriou. „Coupling Source Routing with Time-Sensitive Networking“. In: *2020 IFIP Networking Conference (Networking)*. 2020, S. 797–802 (zitiert auf S. 14, 33).

[9] *Mininet Release 2.3.0*. 10. Feb. 2021. URL: https://github.com/mininet/mininet/releases/tag/2.3.0 (besucht am 11. 03. 2021) (zitiert auf S. 28).

[10] *netem Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/tc-netem.8 (besucht am 14. 03. 2021) (zitiert auf S. 22).

[11] *netns Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/ip-netns.8.in (besucht am 14. 03. 2021) (zitiert auf S. 27).

[12] *red Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/tc-red.8 (besucht am 14. 03. 2021) (zitiert auf S. 22).

[13] *TAPRIO Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/tc-taprio.8 (besucht am 14. 03. 2021) (zitiert auf S. 26).

[14] *tbf Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/tc-tbf.8 (besucht am 14. 03. 2021) (zitiert auf S. 22).

[15] *tc Manpage v5.11.0*. 23. Feb. 2021. URL: https://github.com/shemminger/iproute2/blob/bbddfcec6c32781e5b4915ef4ce6b9b13eed82ef/man/man8/tc.8 (besucht am 14. 03. 2021) (zitiert auf S. 21).

Alle URLs wurden zuletzt am 14.04.2021 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift