Vrije Universiteit Amsterdam

Bachelor Thesis

A performance analysis of TC for high speed, scalable data center networks

Author: Yigit Abaci 2759981

1st supervisor: Daniele Bonetta

daily supervisor: Krijn Doekemeijer

2nd reader: Tiziano De Matteis

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

August 7, 2025

Abstract

Network performance has become the major concern for data centers, while keeping up with the increasing network needs. Performance *Quality of Service* (*QoS*) becomes critical as traffic and infrastructure expand, seeking the need to use tools more efficiently in multi-tenant scenarios. TC, the *Traffic Control* utility in *Linux*, provides a way to control network flows, with the use of different *qdiscs*, performing tasks such as scheduling and providing QoS. However, TC stands with an outdated design and a lack of understanding of its capabilities, with potential limitations. The thesis evaluates the capabilities of TC for performance, scalability and *QoS* in modern data centers, with a setup consisting of a multi-tenant environment using an interface capable of handling 100Gbps, and performance analysis tools – *netperf* and *iperf3*. We firstly conduct a survey and establish our QoS properties as *prioritization*, *fairness* and *work-conservation*. We evaluate the overhead of *qdiscs* in both increasing packet lengths and increasing clients, finding non-negligible overhead for *HTB*. Then we find that *HTB*, in our setup efficiently ensures prioritization by keeping prioritized flows with lower latency. Lastly, we highlight a consistent yield in fairness across all *qdiscs* in our setup. By investigating TC and the *qdiscs*, the thesis supports the development of optimized and more efficient data center infrastructures and systems, providing a better understanding of TC and QoS for modern day adaptation. The benchmarking framework used in this thesis is publicly available via GitHub.

Contents

1 Introduction **1**

1.1 Problem Statement . . . 2
1.2 Research Questions . . . 3
1.3 Research Methodology . . . 4
1.4 Thesis Contributions . . . 4
1.4.1 Conceptual . . . 4
1.4.2 Artifact . . . 5
1.4.3 Experimental . . . 5
1.5 Plagiarism Declaration . . . 5
1.6 Thesis Structure . . . 5

2 Background **7**

2.1 Networking in Data Centers . . . 7
2.2 TC in the Linux Networking Stack . . . 7
2.3 The qdiscs in TC . . . 10

3 Survey on Network QoS and TC Usage **13**

3.1 Method for the Survey . . . 13
3.2 Defining Network Performance QoS . . . 14
3.3 Understanding the Use of TC . . . 19
3.4 Summary . . . 22

4 Experimental Setup **23**

4.1 Hardware . . . 23
4.2 System Configuration . . . 24
4.3 Software . . . 24
4.4 Performance Metrics . . . 25
4.5 Qdiscs . . . 25

5 Measuring TC Performance Overhead **27**
5.1 Method to Evaluate Performance Overhead . . . 27
5.2 Bandwidth and CPU Overhead . . . 28
5.3 Latency Overhead . . . 31
5.4 Summary and Key Findings . . . 33

6 Measuring QoS Properties **35**
6.1 Method for Measuring QoS Properties . . . 35
6.2 Tenant Prioritization . . . 36
6.3 Fairness . . . 37
6.4 Summary and Key Findings . . . 39

7 Reporting Negative Results and Limitations **41**
7.1 Negative Results . . . 41
7.2 Limitations . . . 41

8 Related Work **43**

9 Conclusion **45**
9.1 Answering Research Questions . . . 45
9.2 Future Work . . . 48

References **49**

A Reproducibility **57**
A.1 Abstract . . . 57
A.2 Artifact check-list (meta-information) . . . 57
A.3 Description . . . 57
A.3.1 How to access . . . 57
A.3.2 Hardware dependencies . . . 58
A.3.3 Software dependencies . . . 58
A.4 Experiment workflow and Customization . . . 58
A.5 Evaluation and expected results . . . 58

1

Introduction

Research in computer systems and networks is crucial to sustain the digital economy, which acts as a foundation of the modern day society. For instance, the digital economy accounts for a 60% of the GDP in the Netherlands (1), while enabling over 3.3 million jobs. As the society and the economy becomes increasingly dependent on the digital foundation, infrastructures must address challenges which concern manageability, responsibility, sustainability and usability (1). In the context, performance plays a crucial role for user experience, concerning a wide range of applications such as real-time communication (2, 3), entertainment (4), cloud computing and applications running in data centers. The deployment of large-scale data centers have been increasing significantly over the past decade (5), being used by service providers in the industry such as Amazon (6), IBM (7) and Google (8). Infrastructures need to keep up with the increasing number of users and data transmitted over the network, which are expected to handle *zettabytes* of data in the near future (9).

Data centers are the backbone behind cloud computing, thus their infrastructures need to adapt to the modern day increasing workload needs. These bottlenecks arrive both from the hardware that supports the infrastructures or the software-level mechanisms that assist in controlling the network traffic – such as packet schedulers. In high-throughput environments, controlling the flow becomes essential for network quality. The most dominant choice for data centers when it comes to networking protocols is *TCP* (10). The protocol uses *acknowledgments*, *sequencing*, *error-checking* and *retransmissions*, which ensure reliable and in-order transmission over the network. However, *TCP* is not designed to address *Quality of Service* (QoS) across the network. To fill in this gap, many tools are used to control the flow, which operate on multiple layers of the network stack, shaping the way the packets are transmitted over the network.

Being the dominant choice for data centers (11), *Linux* offers various utilities that can be used to control data flow. TC is one of the most prominent utilities that is used to manage packets at software level, providing control of the network flow, or emulating realistic networks. The tool uses queuing disciplines which are used to achieve control over the packets, including *bandwidth limiting*, *prioritization*, *delaying* and *shaping*, providing a full control over the network flow. Being introduced [1] around the early 2000s (12), TC's design capabilities were designed for the available network needs of its era. As data center infrastructures grow to support more and faster data transfer – utilizing NICs capable of providing more than 100Gbps transfer, while also reaching to more tenants –, understanding and evaluating TC plays a critical role in ensuring performance, scalability and *Quality of Service* (QoS).

We start our research by conducting a survey on *Quality of Service* (QoS), from which we introduce our definitions as a collection of techniques to ensure certain network level expectations and optimizations. We discover that three dominant properties are used on the current *state-of-the-art* use cases; *prioritization*, *fairness* and *work-conservation*. We then, benchmark the capabilities of TC, in modern day applications with high throughput demands (NICs with capabilities up to handling 100 Gbps (13)), via *state-of-the-practice* tools such as iperf3 (14) and netperf (15), where QoS – as well as being defined – needs to be ensured. Within our results we highlight the importance of packet lengths, number of clients, and *qdisc* choices, and how they impact overhead and two of the previously described *QoS* properties (*prioritization* and *fairness*). Further on, we find non-negligible overhead in terms of CPU for *HTB*, we observe latency reduction on prioritized classes when using *HTB* with a prioritization scheme, and lastly observe a consistent yield in fairness for all of the *qdiscs* using our setup.

Lastly, by conducting in-depth benchmarking of TC, the thesis aims to contribute to the constant development and improvement of understanding and using control flow mechanisms in modern day, high-speed and scalable, *Linux* based data-centers. Understanding and evaluating the concept paves way for further optimization of tools and resources to support sustainable and cost-effective data-centers.

1.1 Problem Statement

The constant development and innovation in the industry has introduced many tools and concepts, changing and shaping the understanding of many measurements, including

[1] with the *iproute2* package

QoS, while leaving a gap on a modern-day QoS understanding. **Thus we identify a lack of knowledge on a concrete definition of QoS for networks and how it relates to modern day, high-speed and scalable data center networks (P1).**

While TC can provide reliable control over the network flow, it requires precise configurations which are poorly documented in the documentations for modern day applications. **Thus we identify a lack of knowledge understanding TC configurations and how they map to application demands in modern day, high-speed networks (P2).**

The traffic management utility, TC, was introduced to the *Linux* kernel when the requirements were significantly lower than those required by today's high-speed applications. **Thus we identify a lack of knowledge on the scalability and the overhead of TC for modern use (P3).**

The scalability of such a tool becomes unimportant to control data center networks if it cannot provide the QoS expectations, which are critical for data centers. **Thus we identify a lack of knowledge on how and to what extent TC can achieve QoS needs (P4).**

1.2 Research Questions

Main RQ: *How does TC control performance QoS in the network and to what extent does it provide performance, scalability and QoS when applied to high speed and scalable networks?*

RQ1: *How do we define and evaluate performance QoS in high-speed networks and how does the state-of-the-art use TC to provide these performance QoS properties?*

This question aims to establish an initial understanding and establish the metrics used to help answer the upcoming research questions. Using **RQ1**, we are able to define *performance QoS* in the context of data centers, understand how TC is currently used in the industry and academic research. This way we address **P1** and **P2** directly.

RQ2: *What is the performance overhead in bandwidth, CPU and latency of using TC?*

The first benchmarking question is aimed to establish whether TC can saturate modern fabrics, in a representative environment of a modern high-speed and scalable datacenter, where we aim to demonstrate scalability and cost tradeoffs of TC.

RQ3: *How and to what extent does TC ensure the previously determined performance QoS properties in multi-tenant setups?*

The aim of this question is to apply the discoveries of TC knobs and our QoS definitions from **RQ1** to benchmark a network flow between multiple tenants. This way, we simulate a standardized TCP connection in a datacenter-like environment, which helps us assess the *QoS* properties.

1.3 Research Methodology

The benchmarking will be done using two virtual machines (VMs) operating with a *passthrough NIC*. We will use the ***Mellanox MT27800 ConnectX-5*** [1] over ***SR-IOV*** [2] , providing an environment similar to a modern day high-speed data center. In order to analyze performance of the network flow, we use *iperf* and *netperf* across all the benchmarks. Lastly, we use commonly used techniques in previous research (to be defined under **RQ1**) to measure *work-conservation*, *prioritization* and *fair-sharing*.

1.4 Thesis Contributions

Throughout the thesis, we aim to provide various contributions to the industry and the constant development in networking.

1.4.1 Conceptual

The research provides, through a survey, a definition of QoS in modern day, high-speed and scalable data center networks. We are also to understand the purpose of TC and

[1] Mellanox MT27800 ConnectX-5: a high performance network interface card (NIC), supporting up to *100 Gbps* per port, designed for datacenter-like infrastructures. The interface supports virtualization

[2] SR-IOV: Single Root I/O Virtualization is a standardized method to split a *NIC* over multiple VMs, while allowing optimized performance and minimized overhead

its knobs within the same context. To continue, we are to benchmark the performance, scalability and QoS of TC in a modern day data-center like environments, paving way for future research and development within the field

1.4.2 Artifact

We are to contribute with an open-source benchmarking tool, which is to be used to measure TC performance, scalability and *QoS*. The framework provides a standardized way to evaluate *qdisc* performance, aiding future investigation on TC and *qdiscs*.

1.4.3 Experimental

We are to provide experimental results regarding the use of TC under varying situations that replicate real world data center networks, providing conclusions and insights, and a reproducible baseline for aiding future work.

1.5 Plagiarism Declaration

The content of the thesis is the work of the author. The thesis does not contain any work that has been copied from another source (person, Internet, or LLM).

1.6 Thesis Structure

Firstly, we introduce the required background knowledge and tools regarding QoS and TC. Then we find out how TC is currently being used, or how it can be used, including the specific parameters/configurations/knobs. Further on, we identify network QoS properties, and map them to the said TC specifications, answering **RQ1**, and setting a layout for the benchmarking sections. Then we design and implement a collection of tools that can be used to benchmark the chosen metrics and the related TC configurations. Lastly, the implemented benchmarking tools are used to run the benchmarks to address **RQ2** and **RQ3**, where the results will be evaluated, along with recommendations regarding the scalability and application of TC.

2

Background

2.1 Networking in Data Centers

Networking in data centers plays a crucial role in modern-day data centers, linking infrastructures, services, and consumers. With the increasing number of applications that rely on data centers, networking has become the backbone of performance and quality, providing opportunities for innovation and improvement.

Modern day data centers are designed with the goal of supporting high throughput scenarios while not compromising low-latency network traffic between multiple tenants. To ensure the expected networking needs of the service, many tools have been implemented over the past years. The design of the infrastructures has shifted to programmable data planes, allowing packet processing to be implemented directly in the network. For example, *Sailfish* (16) addresses limitations of traditional implementations by using programmable network switches. Compared to traditional gateways, the algorithm reduces latency by 95% and a significant increase throughput. *Sailfish* has been in use since its deployment by *Alibaba Cloud* (17), handling modern-day traffic loads with ease. Similarly, *vPIFO* (18) is introduced as an architecture for scheduling in modern-day networks. By allowing the traditionally used physical *PIFO* (*Push In First Out*) queues to be virtualized, *vPIFO* achieves a high-speed hierarchical, tree-like scheduling architecture.

2.2 TC in the Linux Networking Stack

In Linux, networking involves different layers managed by the networking subsystem in the kernel. Starting from the lowest layer, the physical network devices use the network drivers to communicate with the kernel. On the other hand, higher levels involve protocols

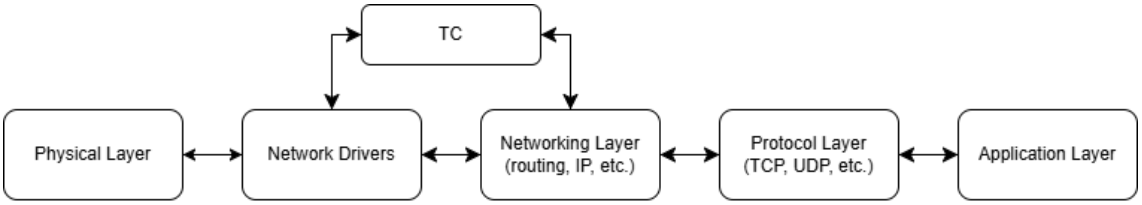


Figure 2.1: Linux networking layers. The figure only includes the relevant layers, the actual networking stack contains additional layers

which provide data transmission, such as the *Internet Protocol* (*IP*) and the *Transmission Control Protocol* (*TCP*). TC, within the layers, stands between the network drivers and the networking layers, where it controls the outgoing or incoming traffic.

TC (Traffic Control) in Linux is a tool that has been introduced as part of *iproute2* [1] and is used to control and shape packets on the network level. TC uses various mechanisms shaped around queuing disciples (*qdiscs*), which are used to control and shape the flow. *Qdiscs* are divided into two categories, classful and classless. Classless *qdiscs* do not support the use classes and filters. The most commonly used example is the FIFO (First In First Out). On the other hand, classes can be used to implement pipelines with filters – such as the u32 – which classify the packet based on the metadata.

TC is a complex tool designed to perform many tasks with many knobs, making it crucial to understand how they function before experimenting with them. We summarize its parameters and their use cases as follows:

- **Queuing Disciplines** are added using ***qdisc***. *Qdiscs* can be extended with child *qdiscs* which provide extra functionality. As an example, the default Linux configuration utilizes a combination of *mq* and *fq_ codel*, where *mq* controls the traffic going through different NICs, and *fq_ codel* controls the packets beforehand.

- **Classes** are defined (only within classful *qdiscs*) by using ***class***. This allows the separation of traffic, where each *class* can have its own shaping and scheduling rules.

- **Controlling the Bandwidth** can be done by using rules such as (but not limited to) ***rate*** or ***ceil*** to limit bandwidth or ***latency*** to set the maximum allowed latency.

- **Policing** limits rate and drops packets that exceed the limit. It is implemented by ***police*** within the TC parameters.

[1] *iproute2* is a collection of tools to monitor and modify networking in the Linux kernel

- **Network Emulation** can be implemented – using **netem** – with tools such as ***delay***, ***loss*** to simulate packet loss, ***duplicate*** to introduce duplicate packets, ***corrupt*** for packet corruption and ***reorder***.
- **Classification** can be done using *filter* with different filtering mechanisms such as ***u32*** for header-based classification, ***fw*** for firewall mark-based classification and ***bpf*** to classify packets with *Berkeley Packet Filters*.

For clarification, we give an example of classful queuing, using *qdiscs*, classes and filters. For this purpose we use *HTB*, where we create a hierarchical structure with a prioritization scheme:

1. **Adding the *qdisc***, where unmatched traffic goes to the class 1:20, set by `default`

   ```
   tc qdisc add dev NIC root handle 1:  htb default 20
   ```

2. **Creating a root class** with a total bandwidth set by the `rate`

   ```
   tc class add dev NIC parent 1:  classid 1:1 htb rate RATE
   ```

3. **Creating (two) child classes** with different priorities – set by `prio`, where the lowest number gets the highest priority – and guaranteed bandwidth – set by `rate` and `ceil`.

   ```
   tc class add dev NIC parent 1:1 classid 1:10 htb rate RATE ceil RATE prio 1

   tc class add dev NIC parent 1:1 classid 1:20 htb rate RATE ceil RATE prio 2
   ```

   In this example we set the class `1:10` to have higher priority than the class `1:20`.

4. **Assigning a *qdisc* for each class**, where we use a *pfifo* (for simplicity)

   ```
   tc qdisc add dev NIC parent 1:10 handle 10:  pfifo

   tc qdisc add dev NIC parent 1:20 handle 20:  pfifo
   ```

5. **Filtering flows into classes** based on port number, using `u32`.

   ```
   tc filter add dev NIC protocol ip parent 1:0 prio 1 u32 match ip dport PORT flowid
                                        1:10

   tc filter add dev NIC protocol ip parent 1:0 prio 1 u32 match ip dport PORT flowid
                                        1:20
   ```

2.3 The qdiscs in TC

TC includes various *qdiscs* which are implemented to be used in various different scenarios. We summarize the intended use cases and their characteristics as the following:

- **FIFO-based *qdiscs*. Pfifo** is a simple FIFO queue with a fixed packet limit, making it simple and effective. Similarly, **bpfifo** can be utilized to control limits in terms of byte instead of packet count, enabling byte-level enforcement. Lastly, **pfifo_fast** implements 3-band prioritization depending on the TOS [1] bits in the IP header of packets. All of the FIFO based *qdiscs* are work-conserving by nature as they transmit packets constantly, while not offering any prioritization or fairness.

- **fq** stands for fair queuing and is the default optimized choice for the kernel. It can also be extended with the *CoDel* (Controlled Delay) algorithm, which drops packets – in contention scenarios – to reduce latency, using **fq_codel**. These two *qdiscs* are designed to provide per-flow fairness and work-conservation, while not having any prioritization effect.

- **TBF** (Token Bucket Filter) is a simple shaping algorithm which is used to enforce bandwidth needs, where packets follow a token based algorithm. This *qdisc* supports no QoS properties.

- **SFQ** (Stochastic Fair Queuing) is designed to provide fairness using its random per-flow hashing and its underlying queues, while being lightweight in terms of system utilization. *SFQ* provides a hash-based algorithm to provide fairness, and additionally work-conservation, while having no priority scheme available.

- **DRR** (Deficit Round Robin) is designed to provide fair distribution of bandwidth while maintaining simple characteristics and lower load on the system. The *qdisc* provides per-flow fair sharing (as of the round robin algorithm), and also yields work-conserving behavior with no prioritization instrument.

- **QFQ** (Quick Fair Queuing) is a low overhead scheduler with fairness guarantees which is designed to maintain its properties when scaled. The flow-aware algorithm provides strict fairness and work-conservation, while not providing prioritization.

[1] Type of Service (TOS) or Differentiated Services (DS) is an 8-bit field within the IPv4 and IPv6 header, which indicates how a packet should be treated in terms of performance. Using the field, packets can be indicated to minimize delay, maximize throughput, etc.

- **HTB** (Hierarchical Token Bucket) is a more complex *qdisc* when compared to the rest, allowing hierarchical bandwidth shaping and classification, which is aimed to enable precise control and provide QoS properties. These characteristics make it suitable for many situations including scalable networking such as datacenter. The *qdisc* provides all of the QoS properties; due to its bandwidth borrowing algorithm it assists in being work-conservative, due to its bandwidth sharing algorithm it provides fairness and lastly, it supports strict prioritization for classes.

- **MQ** (Multi Queue) is used to control traffic into hardware queues when multiple NICs are presents. The hardware *qdisc* is designed to be work-conservative but does not contain any algorithm that aims to provide prioritization and fairness.

- **CBQ** (class-Based Queuing) is a classful scheduler, designed to divide the bandwidth among classes in a hierarchical scheme. The *qdisc* offers a complex implementation, which is designed to offer prioritization, but offers limited fairness and work-conservation. We discard this *qdisc* due it being complex and obsolete for modern-day use.

3

Survey on Network QoS and TC Usage

In this section we conduct two surveys to answer **RQ1**, ***How do we define performance QoS in high-speed networks and how does the state-of-the-art use TC to provide these performance QoS properties?***. The first survey in section 4.2, defines *network performance QoS* within the context of this thesis, throughout its use within previous research in the networking field. This way we address the first part, ***How do we define performance QoS in high-speed networks?***, of the **RQ1**. On the other hand, the second survey in section 4.3 aims to understand how TC has been used throughout previous research and real-world use cases. This way, we cover the second part, ***How does the state-of-the-art use TC to provide these performance QoS properties?***, of **RQ1**, using the definitions from section 4.2.

3.1 Method for the Survey

Before evaluating a large number of papers that are used to answer the research questions, it is important to select and filter the papers using a structured approach and a relevant criteria. For this reason, we use a combination of a *systematic search* – searching specific databases with specific terms and keywords – and the *Snowball method* – acquiring papers recursively starting from a selection of papers.

The used key-words for the search can be listed as the following: *Quality of Service, QoS, Data centers, Datacenter networks, Cloud computing, Traffic Control, Fairness, Work-conservation, Prioritization.*

Finally, the inclusion and exclusion criteria stands as the following: **(1)** For **section 4.2**, the scope of the papers are to concern only networking in data centers, whereas in **section 4.1** we opt for a wider approach, including networking out of the scope of data centers. **(2)** Storage related, or non networking related works are to be excluded. **(3)** While no exclusion criteria is to be made for the year of the publications, papers standing behind the current *state-of-the-art* infrastructure speeds and sizes are to be excluded.

3.2 Defining Network Performance QoS

The definition of QoS varies depending on the context of its application and how the analyzed system is approached (19). For its use in this thesis, a solid description and explanation should be established, by reviewing its previous definitions in the context of data centers and networking.

Early QoS models such as the *Integrated Services Model* defined in *RFC* (20) have set foundational definitions. The model defines QoS as the ability to ensure performance levels in critical real-time applications with various types of traffic, dividing the core concept into three components; resource reservation, admission control and packet scheduling. These mechanisms, not only provided an initial model for QoS, but have also contributed in preparing a foundation for more scalable approaches. An example of this can be seen in another architecture included in the *RFC*, the *Differentiated Services Model* (21). The specification aims to provide a coarse-grained approach to the QoS problem, solving the scalability issues of the previous work.

In addition to models, QoS is also used as a way to define metrics that a service or an application can be measured under. A common way to approach QoS is to define it as the pure network performance which concerns bandwidth, delay, packet-loss and jitter (22, 23). The approach stands as sufficient for most systems, where the output is of the highest priority.

However, with the increasing size of the infrastructures (3), additional considerations such as the hardware performance metrics – such as CPU overhead – are also taken into consideration. Hardware metrics are crucial in order to optimize performance, especially in shared resource infrastructures such as data centers using virtualized machines.

Lastly, as an indirect metric, the cost of the operation in terms of energy and/or capital has been an increasing concern for many data-centers (24). Energy consumption has been the main concern over the past years, with many research and investment towards reducing

the consumption and the environmental effects (25, 26, 27, 28). When it comes to capital, many companies have been trying to find ways on reducing costs.

For the scope of this thesis, we define *performance QoS* as a collection of the following properties:

- **Work conservation** is defined as being able to keep the resources – as an example, the entire bandwidth – occupied. The property is not a quantifiable property due to many reasons. However, many approaches exists for evaluating work-conservation. As an example, *Popa et al.* in (29) comment on the trade-off between bandwidth and link utilization, while evaluation their data center focused bandwidth allocation framework – *ElasticSwitch*. While they do not aim to quantify work-conservation in their work, they relate per-tenant utilization to work-conservation, and investigate the per-tenant idle capacity. However, such measurements, while they fit to their wok, may not be suitable in scenarios with properties such as rapid changes in traffic, due to the measurement being an average of the utilization – where high average utilization does not always make a system work-conservative.

  Similarly, *Hu et al.* in (30) present a work-conserving bandwidth allocation algorithm, *Trinity*, which guarantees each tenant a minimum bandwidth while redistributing idle bandwidth to active tenants. To provide experiments that support their claim for the work-conserving algorithm, they conduct experiments using *throughput* as a metric. Resulting in a higher (per-tenant and aggregate) *throughput* compared to other algorithms, thus supporting their design, standing by the claim that higher throughput indicates using the capacity more effectively, thus supporting work-conservation.

  On the other hand, *Liu et al.* in (31) present a bandwidth guaranteeing and redistributing algorithm similar to the previous papers. However, during their evaluation, they include *queue occupancy* – the data waiting to be transmitted – in their measurements in addition *utilization* and *throughput*. The *queue occupancy* metric measures the traffic that uses the bandwidth, thus detecting under or over utilization, which identifies work-conserving behavior.

  Another approach to different metrics used can be observed in *Homa* (32), where *Montazeri et al.* present a transport protocol for datacenter networks. While the paper does not contain nor comment work-conservation as is, they use the previous metrics (*utilization* and *throughput*) , from which they suggest an effective approach to resource usage, which aligns with the definition of work-conservation.

In order to have a metric that is both quantitative and suitable for the scope of this thesis, we approach the property as the following: **(1)** The property is to be viewed as a qualitative property, where we can define a system as work-conservative or not. **(2)** We introduce a sub-property, *utilization* which concerns both the per-application bandwidth and the per-application throughput.

This way, we are to discuss a property of a system, supporting with a quantitative metric that aligns with the scope of the experimentation.

Lastly, *Alizadeh et al.* in (33) include metrics such as *queue occupancy* and *link utilization* for evaluation. While they do not utilize work-conservation as a term either, their approach avoids using the entire bandwidth, making the design non-work-conservative – to improve latency – pointing out the trade-offs between utilization and latency. On the other hand, the paper indirectly supports the idea of approaching work-conservation as a qualitative resource. For instance, the intentional headroom introduced by *Alizadeh et al.* in their design improves latency but lowers the utilization. Thus if we rely on *utilization* as the only metric, conclusions that do not fit into the scope of the system could be derived regarding the work-conservation of the system. This way, approaching *utilization* as a supporting metric aids in providing a qualitative description of the system.

- **Fairness** (or Fair-sharing) is defined as the act of equally sharing the resources among different tenants. While the property initially does not appear to be quantifiable, many ways to approach the property exists. *Shi et al.* in (34), discuss the meaning and interpretation of fairness (or fair-sharing) and quantify it using various measures. One of the quantitative indexes that is presented is the *Jain's Fairness Index*, firstly introduced by *Jain et al.* in (35). The index provides 4 critical properties; *scalability*, *boundedness*, *continuity* and *metric independence*. The index can be represented as the following equation:

$$f(\mathbf{X}) = \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{\sum_{i=1}^{n} x_i^2}, \tag{1}$$

Where $\mathbf{X}$ represents a set of resource allocations, and $\mathbf{x}$ represents a single allocation within that set.[1]

[1] Allocation in the context can refer to the allocation of resources such as the *bandwidth*

However, as a downside of the provided measurement, *Shi et al.* argue that the measurement is for the entire system and cannot identify unfairly-treated individual parts of the system. Other metrics such as the *entropy* are also included in the discussion.

In (36), *Alizadeh et al* evaluate fairness for the proposed algorithm, *DCTCP*, by comparing *per-flow* throughput of multiple flows. This way, the authors show that their implementation avoids starvation and provides fairness. While the measurements help identify starvation scenarios, they do not represent a quantitative index but rather metrics that support a qualitative property. In contrast, (37), *Wang et al.* proposes an improved version of *DCTCP*, the *DCTCP-FQ* which includes active and fair queuing methods. Within their experiments, *Wang et al.* once again measure the *per-flow throughput* but also use the previously described *Jain's Fairness Index* over the metric. This way, the authors describe the comparison (of *DCTCP* and *DCTCP-FQ*) within their experiments with a quantitative metric.

Lastly, *Popa et al.* in *ElasticSwitch* (29) utilize only the allocation of per-tenant bandwidth within their experimentation to comment on the fairness of the system.

Nevertheless, for the scope of this investigation, we believe that *Jain's index* provides a sufficient quantitative metric for the *fairness* property and that it can be used for the experimentation. Thus we use the index to measure fairness in the bandwidth of the system and the latency of the individual packets.

- **Prioritization** is defined as being able to prioritize classified flows (whether it is a tenant or an individual packet) over others. Many approaches use *latency* – the average response time – and *tail latency* – the worst-case response time – to comment on prioritization (38, 39, 40), where a higher priority is argued to have a shorter latency (41, 42).

  In *pFabric* (43), *Alizadeh et al.* introduce a flow-prioritized transport design, achieving efficient flow scheduling using priority tags within packets. The paper does not define *prioritization* as a metric but it evaluates different metrics to prove *prioritization*. *Alizadeh et al.* use *Flow Completion Time* (*FCT*) – both mean and tail – within their experiments. This way they prove that smaller flows complete faster than larger ones, confirming the effectiveness of their prioritization algorithm. However, the approach is affected by the size of the flow and thus lacks practicality when real-word scenarios arrive.

PIAS is a prioritization algorithm designed for data center networks that works independent of flow sizes, presented by *Bai et al.* in (44). The algorithm is based on a dynamic approach in contrast to the previously described deterministic algorithm *pFabric*. The paper evaluate its effectiveness via an experiment. Within the experiment, the authors, once again, use *Flow Completion Time* – mean and tail – as a metric to prove that their implementation provides effective prioritization. However, to count for the unknown flow size and the dynamic demotion of packets (to lower priority queues), *Bai et al.* introduce a supporting metric, which measures latency for each priority level. The paper argues that higher priority levels are expected to have lower latency. Thus the authors validate their claim by using the additional metric within their experiments.

To continue, in (45), *Grosvenor et al.* present their implementation of a prioritization system based on classes, combining rate limiting and priority queuing. The implementation isolates latency sensitive flows from bulk traffic in datacenter-like networks, achieving predictable latency for applications with high priority. The implementation is evaluated using *latency* and *jitter* (per priority class). The authors demonstrate the low delay on traffic classified as higher priority in increased (background) traffic scenarios.

On the other hand, *Ren et al.* in (46), conduct experiments to evaluate their implementation of a predictable and low latency messaging system. To show that high priority tenants have higher tail latency, and that their implementation reduces the tail latency. Additionally, a *Service Level Objective* (*SLO*) is defined for the experimentation, and the violation rates of the *SLO* (with and without their implementation) are compared. This way they are to investigate their implementation, not only by measuring (tail) latency – similar to the previous papers – but also by defining a system-specific *SLO* which serves as a tool for comparison.

Another example of quantifying prioritization can be observed in (47). While what *Zuhra et al.* present in the paper – a QoS aware routing protocol designed for *Wireless Body Sensor Networks (WBSN)* – does not directly align with the scope of the thesis – networking in data centers – it does set an example on prioritization measurements. *Zuhra et al.* evaluate their implementation, comparing the *average latency* of different priority levels. They argue that the prioritized classes experiences lower *average latency*. Additionally, the *packet delivery ratio* (*PDR*) is used to argue

that higher priority packets have a higher *PDR*, as they are less likely to be dropped (in high network traffic and congestion scenarios).

In conclusion, we have observed many approaches to measure and argue prioritization quantitatively across various papers. To be used in the evaluation sections, we use *average latency* and the rate of *SLO* violations to quantify prioritization.

The definition and the metrics are to be used throughout the paper, including the benchmarking sections (5 and 6).

The motivation behind choosing certain properties over others is to approach a well established field within a different perspective. As reviewed before, it is clear that in the past, QoS has been defined repeatedly under the same (if not really similar) properties, which are related to pure network performance. By approaching QoS as a collection of *work conservation, fairness* and *prioritization* on top of the already established properties, we widen the effect of the research.

3.3 Understanding the Use of TC

A complex tool such as TC should be studied respectively before benchmarking an reaching to conclusions. Thus, we begin by reviewing how TC works, and how it has been used in previous work.

Bandwidth Enforcement and Rate Limiting plays a key role in virtualized data center-like environments. As an example, the *BwE* framework (48) relies on a large-scale deployment of *HTB*, demonstrating the complexity of TC at scale. The framework is hierarchically structured; while bandwidth and rate allocations are applied on the host-level, the decisions are abstracted on a centralized control level. Within the *BwE* framework, *HTB* acts as a tool to enforce bandwidth allocations (and collect measurements) on the host-level, the lowest level of the *BwE* hierarchy. While TC requires complex monitoring and frequent adaptation, *Kumar et al.* provide a centralized framework that abstracts the tool, allowing global enforcement policies to be applied on the per-host level. The framework underlines the importance of both abstracting and optimizing TC and highlights its configurations, in modern-day, large-scale and performance oriented systems. Similarly, *Zhang et al.* present *Amoeba* (49), a *Deadline-based Network Abstraction* (*DNA*). The abstraction guarantees transfer deadlines by implementing a feedback-based centralized controller, which controls the per-tenant bandwidth. While the controller calculates the per-tenant rate, each tenant is enforces to apply the rate received by the controller using

HTB. The *qdisc* is used on end-host servers which are to initiate transfers between data centers. Each transfer within the system is classified into a class within the two-level *HTB* hierarchy, where the root class classifies outgoing packets, and the children enforcing per-flow rates. This way, *Zhang et al.* provide a way to dynamically enforce rate allocations controlled by a centralized controller, and achieving high utilization, where flow deadlines are met. Additionally, the authors highlight that, the CPU overhead is negligible (approximately 3%) despite of the fine-grained bandwidth control *HTB* provides, making it practical and scalable for its application within modern-day, high-speed and scalable data center infrastructures.

Traffic Shaping and Scheduling. *Carousel* (50), a scalable traffic shaping and scheduling framework, which uses TC as a supporting mechanism for packet shaping. *Saeed et al.* use the *Hierarchical Token Bucket* (*HTB*) within their implementation to shape the traffic by queuing packets, waiting for available bandwidth. While the *qdisc* does not perform scheduling exclusively, it plays a critical role in support the rest of the implementation which is responsible for packet scheduling. Each leaf class of the *HTB* corresponds to a specific *BwE* task flow group, which enables in, not only per-group (coarse-grained) rate limiting, but also per-group traffic shaping – unlike the previously commented per-tenant (fine-grained) implementations. The grouping of multiple flows paves way to a more scalable approach, due to a reduced number of *qdiscs*. This way the paper highlights the trade-off between granularity and scalability, and addresses the potential scalability limitations of *HTB*, utilizing it as a coarse-grained supporting mechanism.

QoS enforcement, specifically **prioritization**, can be implemented in TC via the *prio* queuing discipline (*qdisc*). An example of the use can be observed in *QJUMP* (45), where *Grosvenor et al.* present their implementation which uses TC to isolate latencies of differently prioritized classes. The paper implements strict traffic separation among classes, which results in low latency. With their implementation, the authors underline the capabilities of TC in data-center like scenarios where enforcing network properties – such as *prioritization* – becomes an objective, while outperforming similar mechanisms. To continue, *Waskiewicz Jr et al.* in (51) discuss various features of TC with QoS enforcement being the goal, within *Data Center Bridging* (*DCB*) scenarios. The paper discusses the use of TC *qdiscs* such as *prio*, *multiq* and *cbq*, the use of the *u32* classifier and the *skbedit* functionality. Firstly, the authors suggest that, in *DCB* scenarios, *prio* – the commonly used priority *qdisc* – should be replaced by *multiq* – the multi queue *qdisc*, which applies a *round-robin scheduling* algorithm. The presented approach provides true separation across queues in the hardware, instead of the software-based priority scheme *prio* applies. The

authors argue that true separation only exists while using *multiq*, as in *prio*, a lower priority packet can still delay a higher priority packet, due to them using the same hardware queue. The paper acknowledges *cbq* – a classful hierarchical *qdisc* – as a potential candidate for the implementation. Despite the potential prioritization and shaping capabilities of *cbq* the papers highlight it as complex to configure and less predictable when compared to *prio* or *multiq*. Secondly, the paper uses the *u32* classifier to classify the packets into classes. Later on, these packets' metadata is accessed and modified by the *skbedit* functionality, setting the target queue for the packet.

Network Emulation and Simulation. Many approaches to TC concern the use of *qdiscs* in like *netem* (Network Emulation) or *HTB* to provide realistic simulations for data center like networks. For example, *netem* can be used to introduce delay and packet loss to flows, as observed in (44). *Bai et al.* evaluate *cross-datacenter congestion control protocols* by using *netem* to control the round-trip times. Similarly in (52), *Zhang et al.* investigate the behavior of *Explicit Congestion Notification* (*ECN*) – a network congestion signaling mechanism which avoids dropping packages, for protocols such as IP, TCP and UDP – under a fixed *Round Trip Time*(*RTT*) value. The paper adds delay within the traffic to emulate asymmetric *RTT* behavior. This way, the paper demonstrates the instability of *ECN* under the non-fixed *RTT* scenario, while highlighting how TC can be used to emulate realistic models. Lastly, in (53), *Becker et al.* analyze the accuracy and scalability of TC's *netem* for large-scale and realistic emulation. The paper evaluates the overhead of *netem*, as well as the accuracy, when emulating latency and packet loss. They highlight that *netem* reaches bottlenecks – in terms of jitter and CPU – when scaled. Lastly, as another example, in (54) *Ali et al.* integrate TC in a virtualized environment to control network characteristics – such as the bandwidth and the delay – to emulate network behavior, using the *qdiscs*. While the paper does not specify the *qdiscs* and knobs used, their implementation of the *Load Balancing* algorithm is evaluated realistically using TC.

Finally, TC not only stands as a research tool, but also is being used as a component of complex, state-of-the-art virtual networking mechanisms. Open vSwitch (55) – "a multi-layer, open source virtual switch" – uses TC in its implementation, enabling fine-grained control, supporting features such as QoS enforcement, tunneling and hardware offloading, acting as a key component to modern day, software-defined infrastructures. To start with, the previously seen *HTB* is used to apply per-port limits, enforcing per-tenant bandwidth limits, as well as using *police* for policing. Additionally, the implementation uses *flower* – a knob which allows per-packet control based on flow – to perform hardware offload-

ing, reducing the CPU overhead. The use of TC in such state-of-the-practice systems demonstrates the importance and the scalability potential of the tool.

3.4 Summary

In **section 4.1** we start by surveying papers from which we define our QoS properties. We start by exploring papers that use different metrics to argue *work-conservation*, and conclude that it is to be supported and argued using *utilization* of *per-application bandwidth* and *per-application* throughput. To continue, we observe that *Jain's Fairness Index* is a standard way to quantify fairness in data center networks. Lastly, we explore into metrics that support prioritization in papers related to data center networks, from which we conclude that *average latency* is to be used for further evaluation within the thesis. This way we address the first part of **RQ1** directly, answering ***How do we define performance QoS in high-speed networks?*** with the metrics found to be used in evaluation.

In **section 4.2**, we explore the different deployments of TC in data center networks. We find out that TC covers a wide range of applications within data center-like environments. We find out that the use of TC in previous research falls under four main categories: *Bandwidth Enforcement and Rate Limiting*, *Traffic Shaping and Scheduling*, *QoS enforcement*, and lastly *Network Emulation and Simulation*. While some of the papers cover more than a single category, the section aims to clarify the use of TC in an efficient manner. Additionally, we find out that *HTB* was among the most commonly used features of TC, playing a critical role under all the mentioned categories. This way we address the second part of **RQ1**, answering ***How does the state-of-the-art use TC to provide these performance QoS properties?***.

4

Experimental Setup

Within this chapter, we define and specify the setup and the configurations on which the experiments will be held, and provide a structured evaluation plan.

4.1 Hardware

The datacenter-tenant simulation throughout the evaluation utilizes 2 separate nodes, which share a large portion of the properties and configurations. The host devices consist of the following properties:

- a 20-core 2.00Ghz Intel(R) Xeon(R) Silver 4416+ CPU with a single socket,having 20 physical cores. For the node that acts as the server, hyperthreading is enabled.
- 256GB of DDR4 DRAM
- a Mellanox ConnectX-5, capable on operation at 100 Gbps.

The benchmarks run on virtual machines (VM) that have the following properties:

- a 10-core 2.00GHz Intel(R) Xeon(R) Silver 4416+ CPU with one socket, 10 physical cores and 1 thread for each core. Using a single NUMA node.
- 8GB of DDR4 RAM allocated to the VM from a single NUMA node.
- a Mellanox ConnectX-5 backed virtual NIC, passed through using SR-IOV, with 9000 byte MTU (jumbo frames)

4.2 System Configuration

The host machines have the following software specifications:

- Ubuntu 22.04.4 LTS
- Kernel version 5.15.0-124-generic for node8 and 5.15.0-140-generic for node9
- QEMU version 9.0.1

The guest VMs have the following software specifications:

- Ubuntu 20.04 LTS
- Kernel version 5.12.0+

As demonstrated by *Cai et al.* in (56), TSO and GSO can be used to reduce CPU overhead on high-speed networks with greater packet lengths. The paper provides evidence which imply that with packet lengths smaller than 4.5KB, using TSO or GSO provides no additional benefit. We confirm this on our setup by conducting a micro-benchmark (using *iperf3*) that assesses a small packet length – 1KB. With the results from the experiments, we confirm that enabling the optimizations do not provide any benefit. However, for consistency and to approximate real-world datacenter setups (57), we keep TSO and GSO enabled. Additionally we disable LRO and GRO to facilitate per-packet qdisc evaluation, and per-packet inspection (57).

4.3 Software

To generate benchmarks and obtain network metrics we use the following tools:

- We use *iperf3* (version 2.10.0) to generate TCP flows and simulate concurrent TCP connections.
- We use *netperf* (version 2.6.0) in TCP_RR mode, allowing us to measure round-trip latency.

To measure system metrics, we use the following tools throughout the evaluation:

- We use *pidstat* (version 12.2.0) to obtain per-process CPU statistics
- We use *sar* (version 12.2.0) to monitor system CPU statistics, allowing us to monitor per-core activity.

Lastly we utilize custom bash scripts to automate the experimentation process and *python* scripts to visualize the data, using *pandas* and *matplotlib*. With the use of these tools and scripts, we are able to form a reproducible and consistent framework for evaluating *qdisc* performance under different conditions.

4.4 Performance Metrics

Throughout the evaluation part we use different metrics for different properties and measurements defined in our survey (section 3), which can be listed as the following:

- **Bandwidth** is measured in `Gbits/s` or `Mbits/s` directly obtained from the log files of the benchmark generators. We obtain both per-flow and per-system bandwidth to evaluate the *qdiscs* both for overhead and for the QoS properties.

- **CPU** utilization is measured as a percentage within the sar and pid files. To ensure consistency, we observe both the per-process statistics and the system wide statistics. We observe the `%wait`, `%usr` and `%system` columns within the files, which help us understand the results.

- **Latency** is measured as the average latency of the individual packets in terms of micro-seconds, within the first evaluation. Additionally, to evaluate the properties, we introduce a *SLO violation* scheme which allows us to comment on the properties such as prioritization. This way we bridge our findings from **RQ1** and use them as key metrics within the evaluation. We calculate latency via the `throughout` field within the generator logs, which we take the inverse of.

This way, we combine our main findings from the survey on section 3, to connect the different parts of the thesis and establish metrics to be used in section 5 and 6 during the benchmarks, to comment on the overhead and the QoS properties of various *qdiscs*.

4.5 Qdiscs

We chose a varying range of *qdiscs*, obtained both from the survey and additional ones which help extend the scope of the thesis. We list the *qdiscs* with brief descriptions as follows:

- ***mq + fq_ codel***, being the default selection within the Linux kernel, we include it in the research as a baseline comparison. The kernel utilizes **mq** in scenarios where multiple NICs are present. However in our setup, even though multiple NICs are deployed, only a single NIC is utilized by the VM, thus making the effect of **mq** negligible. Within this setup, **fq_codel** acts as the main *qdisc*, controlling the flow. Nevertheless, we keep the default setup untouched to have a relevant baseline. Lastly, we discard *fq*, *fq_pie* and *SFQ* from the experiments due to the their similarities to **fq_codel**.

- ***pfifo*** is included in the research as a minimal effect *qdisc* allowing us both to sanity check if needed and also highlight the performance of the other qdiscs, especially when it comes to providing QoS properties. We do not include the other FIFO based *qdiscs* (*bpfifo* and *pfifo_fast*) due to their similarities.

- ***HTB*** has been established as the most commonly used *qdisc* – in data centers – in section 3. In our research it plays a critical role to observe the trade-offs between performance and QoS ensuring.

- ***QFQ*** has a strong theoretical approach to QoS properties, while maintaining optimized performance. However the scalability remains unclear, thus we include the *qdisc* in the thesis.

- ***DRR*** is designed to balance performance with QoS enforcement, playing a key role compared to other *qdiscs*, and thus is included within the scope of the thesis.

This way, we cover a varying range of qdiscs which are applicable to the context of modern-day data-centers. We solely include a selection of the *qdiscs*, from the most simple classless qdiscs, to the most commonly used and more complex classful qdiscs. The selections covers all of the relevant *qdiscs* in terms of data-center and QoS relevance, while discarding the redundant ones.

5

Measuring TC Performance Overhead

Within this section we perform benchmarks to evaluate the performance of the selected qdiscs, with the goal of directly answering **RQ2**, *What is the performance overhead in bandwidth, CPU and latency of using TC?*

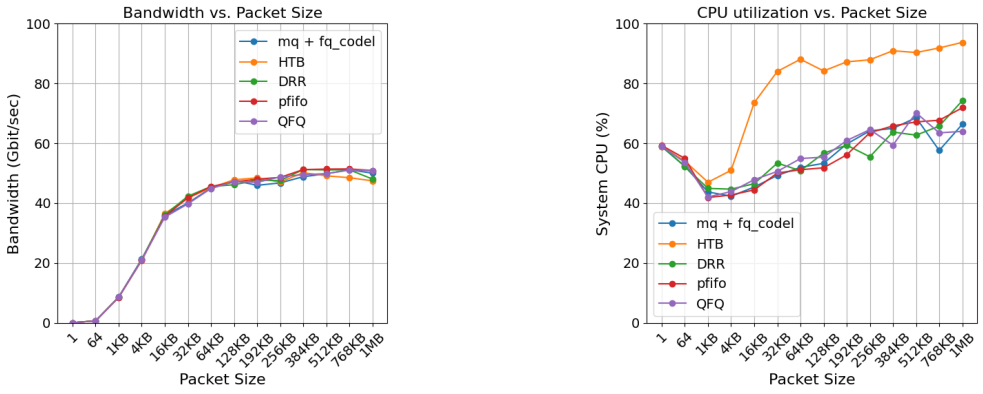
5.1 Method to Evaluate Performance Overhead

To answer **RQ2**, we evaluate the overhead of the qdiscs in terms of bandwidth, CPU utilization and latency. We achieve this by dividing the experiments in two sections, 5.2 and 5.3.

In section 5.2, we conduct 2 series of experiments using *iperf3*. Firstly we focus on the increasing packet length, which is set by the `-l` option within the tool. As the values, we increase the packet length, starting from the minimum value (1 byte) to the maximum (1 megabytes). This way, we are to: **(1)** Saturate the qdiscs and observe the saturation point and **(2)** Find reasonable packet lengths to use within the upcoming experiments.

Then we continue with scaling up clients, where we deploy separate instances of *iperf3*, with packet lengths obtained from the previous experiment, which help us link the two experiments and comment on the scalability at an optimized point. We increase the number of clients with the goal of observing the scalability point of the qdiscs in multi-tenant scenarios, replicating real-world datacenter scenarios.

We obtain results from the *iperf3* log files, which calculates the bandwidth every second, giving both the individual and the average of the whole experiment. Additionally the log includes the number of retransmissions which help us justify cases where unexplainable



(a) (b)

Figure 5.3: (a) Bandwidth vs. Packet Size, (b) CPU Utilization vs. Packet Size.

bandwidth values occur. We obtain CPU values from the log files of *pidstat* and *sar*, giving us adequate per-process and system wide information, separated into `user`, `wait` and `system`.

In section 5.3, we follow the same approach and first saturate in terms of packet length and then in terms of client count. However, we use *TCP_RR* in *netperf* as our benchmark generator, as – unlike *iperf3* which measures bandwidth directly – *TCP_RR* measures round-trip latencies of packets, allowing us to observe the latencies of individual flows. This way we can comment on the latency of the qdiscs under similar conditions.

In summary, we utilize a planned approach with the goal of measuring the pure overhead introduced by the qdiscs while comparing them to each other, when no shaping or QoS scheme is deployed.

5.2 Bandwidth and CPU Overhead

We start by investigating the effects of the changing packet size on the bandwidth and the CPU utilization. For this experiment we hypothesize that the *qdiscs* will perform similarly in terms of bandwidth but show difference when it comes to CPU utilization, especially with more complex ones such as *HTB* and *QFQ*. Since we increase the packet length, we hypothesize that the bandwidth will increase until reaching the saturation point of the *qdiscs*.

In figure 5.3 (a) we observe the increasing bandwidth length and highlight the saturation point, which sits at 128KB for all of the qdiscs, underlining the similarities. On the other hand, with graph (b), we observe a significant difference we observe that ***HTB***

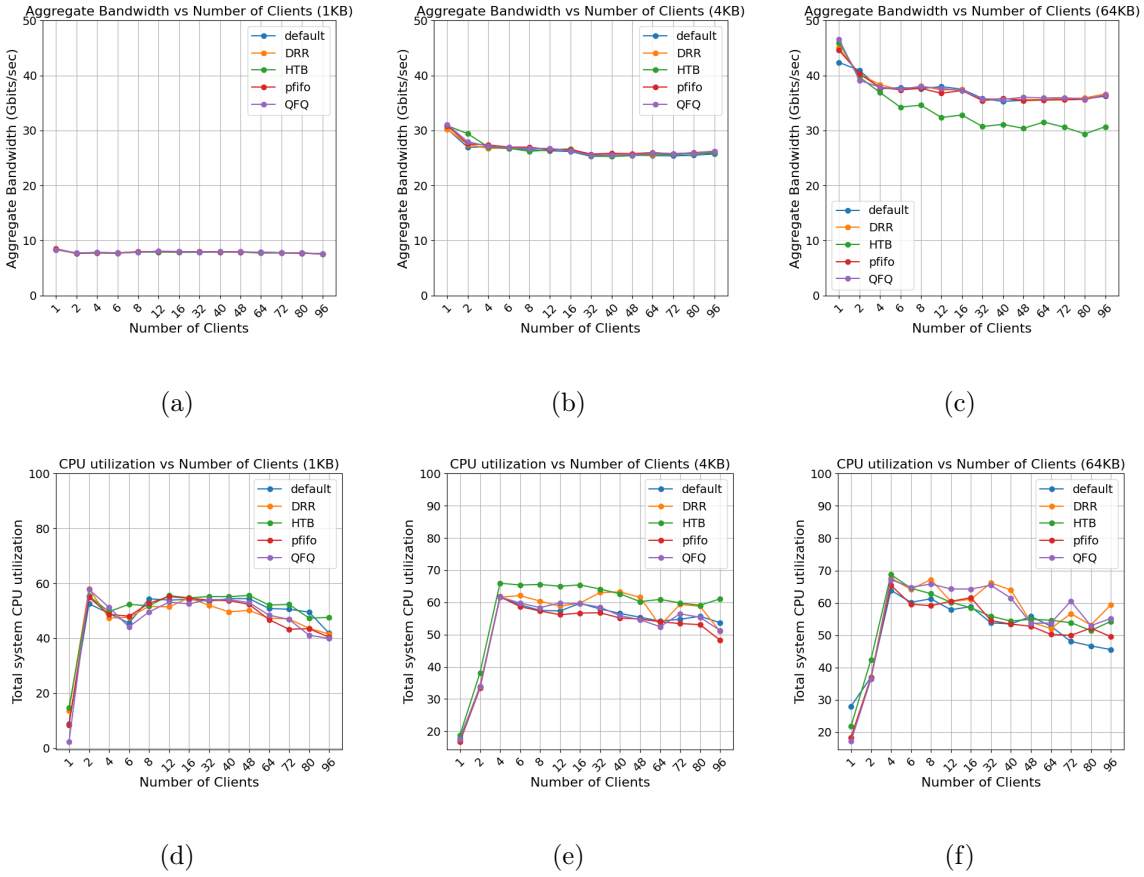


Figure 5.10: CPU performance under scaling clients with packet lengths of 1KB, 4KB and 64KB.

introduces between 5% to 30% CPU overhead when compared to other *qdiscs* for packet lengths greater than 1KB (KF1). This is due to *HTB*'s internal per-byte token algorithm (which we discuss further using the following experiments), justifying the increase in CPU utilization. This way, we highlight the expected CPU cost of *HTB* and how packet length selection controls the CPU overhead.

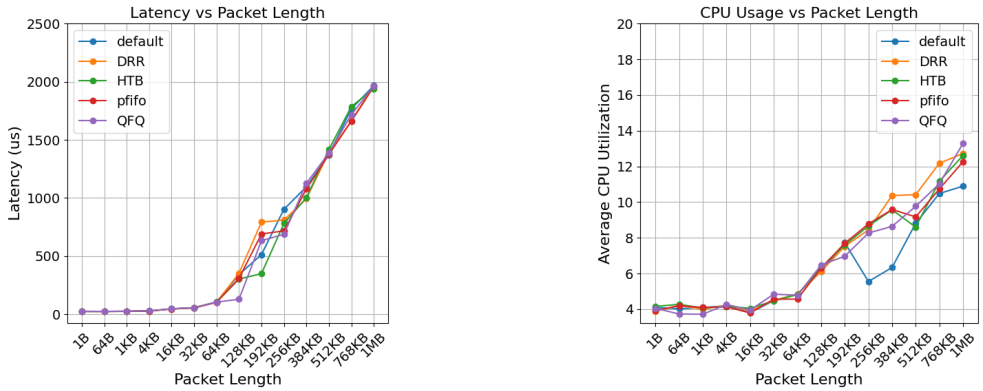
Secondly, we conduct a saturation of the client number, where we keep the packet length controlled and increase the client instances. We hypothesize that simpler qdiscs such as *pfifo* will experience a decrease in terms of bandwidth, especially on higher client counts, whereas more complex ones such as *HTB* will outperform the others, while as a consequence, utilizing more resources. For this we pick 3 different packet lengths: 1KB – representing a small workload, 4KB – representing a moderate workload and 64KB – representing an inflated workload. Using these workloads, we do not over saturate our experiment and control the variable further and provide more accurate conclusions.

We firstly compare (a), (b) and (c) in figure 5.10 observing that larger packet lengths are less optimal, in terms of CPU overhead, for *qdiscs* – which we have demonstrated previously – within multi-tenant situations and show a decrease in bandwidth as the number of clients increase.

Using the 1KB experiment in (a) , we observe a flat line for bandwidth and moderate to high (between 60 and 70%) CPU utilization. As *qdiscs* behave similar under the conditions, we conclude that **the bandwidth is limited by packet processing further within the networking stack rather than *qdisc* logic (KF2)**. We support this by observing graph (d), where *HTB* does not show a significant utilization (whereas it was to be CPU heavy within the previous experiments) and leads us to hypothesize that the overhead rises from packet processing or beyond – which we cannot assess with our setup.

To continue, we observe a diverging bandwidth graph on graph (b) with a 4KB packet size. The CPU graph, (e) of the experiments exposes a consistent overhead (around 5%) of using *HTB*, as well as *DRR*, which shows a higher utilization overall compared to the other *qdiscs*. Hereby, we observe that ***classful qdiscs* introduce 5% more CPU overhead caused by underlying logic (KF3)**. In contrary to the previous experiment where we argued that the utilization was beyond *qdiscs*, we now observe a difference in the graphs, thus concluding that the observed overhead is indeed caused by the *qdisc* logic when 4KB workload is used.

Using even larger packet lengths, 64KB, we observe *HTB* experiencing a drop in bandwidth, whereas the CPU utilization does not contain a proportional effect. Starting from 4 clients, *HTB* drops between 5 to 7 Gbits/sec (between 12 and 17%) in bandwidth. As the packet length grows, the CPU is utilized less for system calls. However, *HTB* fails to keep up with other *qdiscs*, not because it is heavy on the CPU, but due to its token bucket processing logic. *HTB* deploys a token bucket algorithm where tokens are consumed per-byte and not per-packet. This leads to a bottleneck; as the packet length grows, packets consume more tokens (as they are larger in terms of bytes), and *HTB* fails to keep up. To summarize, **we find an internal *HTB* bottleneck, where token consumption grows due to the increased number of bytes after 4 clients and 64KB packets, where *HTB* fails to keep up, leading to a 12 to 17% decrease in bandwidth (KF4)**. On the other hand, using the same graphs we observe that ***pfifo* and the default configuration (*mq+fq_ codel*) maintain bandwidth with a stabler utilization, demonstrating better scalability (KF5)**.



(a) (b)

Figure 5.13: (a) Latency vs. Packet Size, (b) CPU Utilization vs. Packet Size.

5.3 Latency Overhead

We firstly conduct an increasing packet length experiment to assess the baseline performance under increasing packet lengths and sanity check our results, where we hypothesize that the latency will increase as the packet length increases.

In figure 5.13 we see the hypothesized rise in latency in graph (a). We firstly observe that the latency follows an indistinguishable increase until 64KB. Then we observe a steeper increase in latency between 64KB and 256KB for *pfifo*, *DRR* and *QFQ*, while a more gradual increase is observed in *mq+fq_codel* and *HTB*. However after 256KB we observe that all *qdiscs* converge to the same line, thus allowing us to conclude the effect of packet size on latency. At larger packet lengths we observe bottlenecks caused by limitations beyond *qdiscs* – such as network interface limitations – whereas in smaller packet lengths, the overhead in latency is caused by the underlying logic of the individual *qdiscs*.

We then observe varying CPU utilizations on graph (b). Where we observe that the default setting, ***mq+fq_codel* experiences an improvement of 4% of utilization after 192KB (KF6)**, while the rest follow a similar pattern. However, we observe that *DRR* experiences additional 1 to 2 percent overhead in terms of CPU when compares to the others, introduced by the per packet decision logic. Lastly, we observe that *HTB* and *pfifo* follow a similar pattern. This is due to the fact that *HTB* is deployed with *pfifo* as the underlying child *qdisc*.

Followingly, we continue with scaling up the number of clients (flows), where we hypothesize that the latency will increase when the number of clients increase due to contention and retransmissions. Our hypothesis is that simple *qdiscs* will outperform more complex

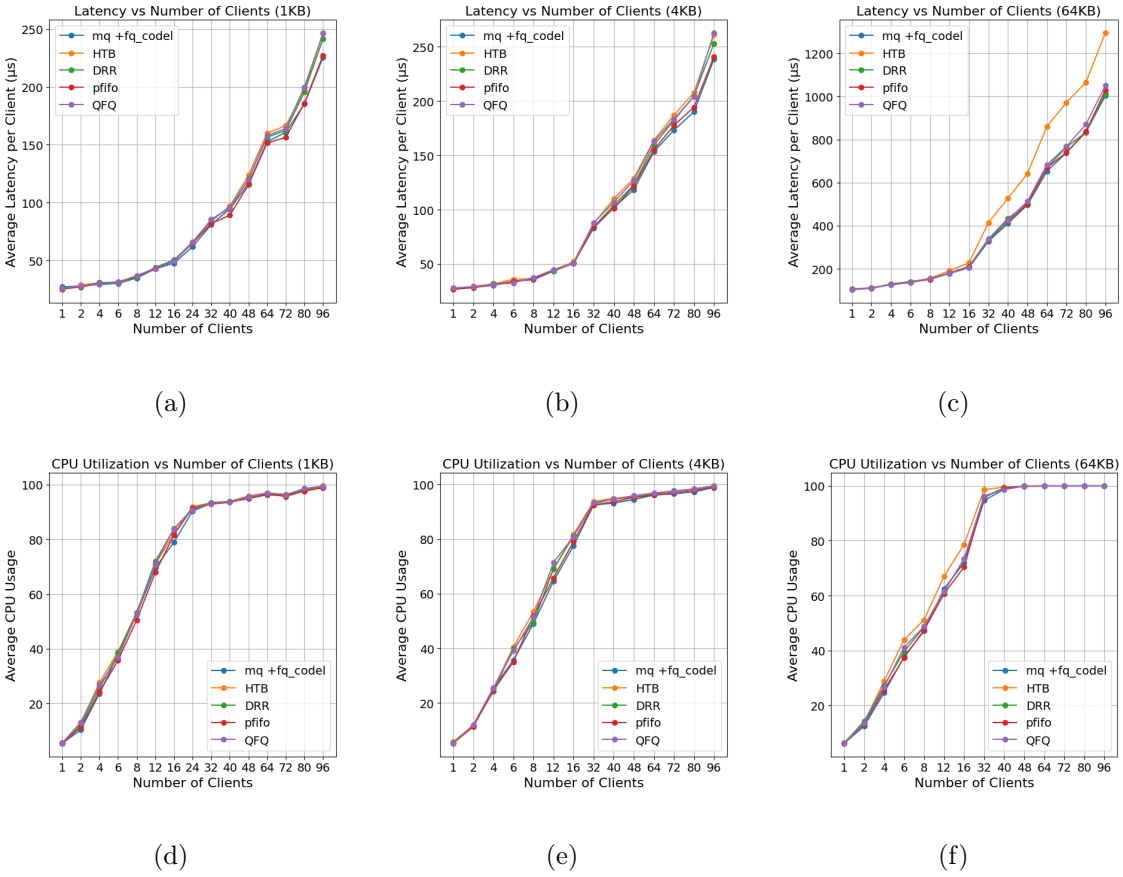


Figure 5.20: Latency performance under scaling clients with packet lengths of 1KB, 4KB and 64KB.

qdiscs as we assess the *latency*, where *qdiscs* that have less internal processing to perform send packets faster, leading to lower latency.

In figure 5.20 (a), we observe the hypothesized rise in latency – with 1KB packet length–, where *qdiscs* perform close to identical. However, using graph (d), we observe that at 96 clients we reach 100% CPU utilization – a system bottleneck. Thus we only confirm that, with smaller packets, *qdisc* logic's effect on latency becomes negligible.

To continue, in graph (b), **when we use a packet length of 4KB, *qdiscs* do not show a clear divergence in terms of latency nor CPU utilization (KF7).**

Lastly, in graph (c), we observe a steep increase in latency of around 20% – which also starts earlier – when using *HTB* – which is also reflected in the CPU utilization in graph (f). This way we support our comments in section 5.2, confirming **KF3** and **KF4**. Lastly we conclude that the saturation point arises early, where the system bottlenecks (after 40 clients).

5.4 Summary and Key Findings

We list the key findings within the experiments performed in section 5.2 and 5.3 as the following:

- **KF1**: We observe between 5% and 30% CPU overhead when comparing *HTB* to the rest of the *qdiscs*, in single-tenant scenarios with packet lengths larger than 1KB.
- **KF2**: We find that the bandwidth is limited by packet processing further in the stack rather than *qdisc* logic when using packet lengths of 1KB.
- **KF3**: We observe that *classful qdiscs* (*HTB* and *DRR*) introduce 5% additional CPU overhead when using a moderate load of 4KB.
- **KF4**: We observe a token consumption bottleneck in *HTB*, after 4 clients and 64KB of packets, leading to between 12% and 17% decrease in bandwidth.
- **KF5**: We observe better scalability due to stabler utilization when using *mq+fq_ codel* and *pfifo* within our three different packet lengths.
- **KF6**: We find an improvement of 4% in CPU utilization when using *mq+fq_ codel*, when using packet lengths greater than 192KB in our latency-sensitive setup.
- **KF7**: We observe no difference in terms of latency or CPU utilization among *qdiscs* when using 1KB and 4KB packet lengths.

using our findings, we answer **RQ2**, *What is the performance overhead in bandwidth, CPU and latency of using TC?* as follows: We find that TC introduces non-negligible overheads, especially with *classful qdiscs* such as *HTB* and *DRR*, highlighting the cost tradeoffs and bandwidth limitations under contention in multi-tenant scenarios.

6

Measuring QoS Properties

Within this section we perform benchmarks to evaluate the performance of the selected qdiscs, with the goal of directly answering **RQ3**, *To what extent does TC ensure the previously determined performance QoS properties in multi-tenant setups?*.

6.1 Method for Measuring QoS Properties

To assess the different properties, we divide the experimentation into 2 different section, each assessing a single property. This way, we isolate properties and approach the research question with a structured plan.

In **section 6.2** we evaluate prioritization using multiple instances of benchmark generators which are bind to different priority classes using `filters`. We summarize the prioritization experiments as the following:

- **P0**: We use this experiment to establish a controlled experiment environment, by using 2 classes – only for classful qdiscs – with no priority applied. This way we are to observe a difference when the priority scheme is applied, validating our experiment. We use a single instance of *iperf3* and a single instance of *netperf* – our latency sensitive flow – to measure the baseline performance of our tools using the setup.
- **P1**: We utilize 2 classes, where one gets higher priority than the other, and we observe the baseline prioritization of the *qdiscs*. We utilize the higher priority class as our latency-sensitive measuring class, which runs *netperf* in `TCP_RR` mode. On the other hand the lower priority class runs *iperf3* – for more precise flow control – and generates background traffic. Then we scale up the number of flows within the lower priority class to observe wether the *qdisc* provides the same prioritization

under increasing background traffic. We do this by increasing the instances of *iperf3* flows assigned to the lower priority class.

We apply **P0** and **P1** solely for *HTB*, the only *qdisc* among our selection on the previous section that supports strict prioritization using classes.

In **section 6.3** we address the fairness of the *qdiscs*, using two experiments, which we summarize as follows:

- **S0**: We use our experiment data from section 5, and evaluate the fairness of *qdiscs* under ideal conditions – concurrent flows and equal packet lengths, with increasing clients.
- **S1**: We introduce flows with different demands, which helps us evaluate actual performance under varying realistic scenarios. For this reason, we vary the packet length of the flows, where 50% of the flows get the base packet lengths, 25% less than the base length and the rest of the 25% more than the base length.

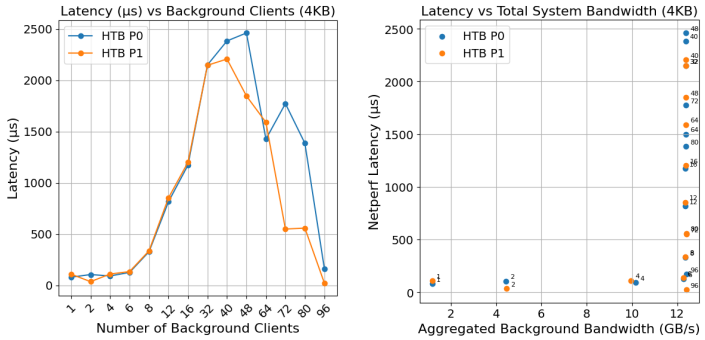
Lastly, we base all our experiments which run on a single packet length, using 4KB, as previously – on chapter 5 – we have observed that smaller packet lengths (such as 1KB) did not cause sufficient traffic to make a difference among *qdiscs*. Similarly, with greater packet lengths (such as 64KB), *qdiscs* suffered from lack of system resources (CPU) and/or extreme contention causing inaccurate conclusions.

As to conclude our method, using our experiments **P0**, **P1**, **S0** and **S1**, we assess two out of the three *QoS* properties defined in our survey on chapter 3, covering the *qdiscs* evaluated on the previous chapter.

6.2 Tenant Prioritization

We hypothesize that, when *HTB* is deployed without rate limits, using only class priorities, assigning a lower numerical class (`prio 1`) to the latency sensitive *netperf* will reduce the latency under contention (greater number of clients), when compared to the setup which utilizes no priority scheme. We relate this to *HTB*'s strict priority mechanism, which will ensure that packets that belong to higher priority classes (lower numerical values) are dequeued before the rest of the packets (from lower priority flows).

In figure 6.3 (a), we observe a sharp increase as the background clients increase, peaking around 48 clients, for both configurations. At lower number of clients, the performance of the setups are identical, as background load stands insufficient to saturate the *qdisc* and



(a) (b)

Figure 6.3: Comparison of **P0** and **P1** for *HTB*, with 4KB packet length

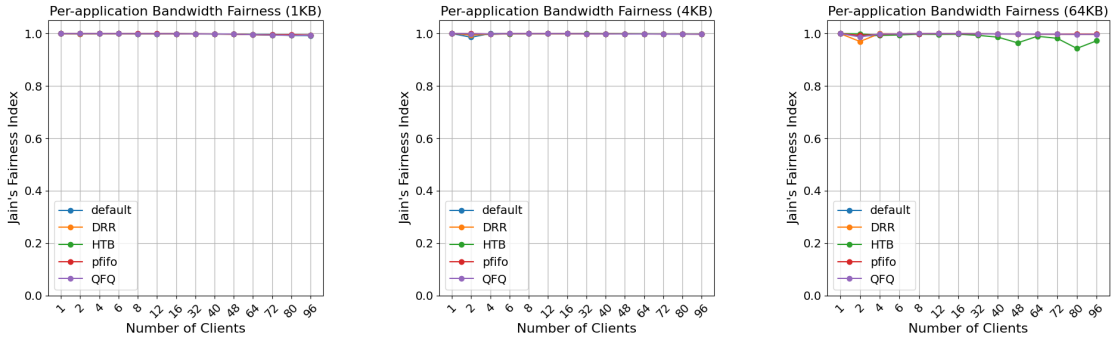
cause contention. However, once we increase the number of clients – beyond 40 clients –, the priority enabled setup (**P1**) experiences an improvement in latency for the higher priority class. We observe a **consistent reduction, ranging from 10% (at 40 clients) up to 350% (at 72 clients) (KF8), when compared to P0**.

Further on, graph (b) support our findings, showing a relation between the latency and the bandwidth. Here, we confirm that the decreased latency does not come at a cost of decreased bandwidth, confirming the strict priority effect of using *HTB*.

6.3 Fairness

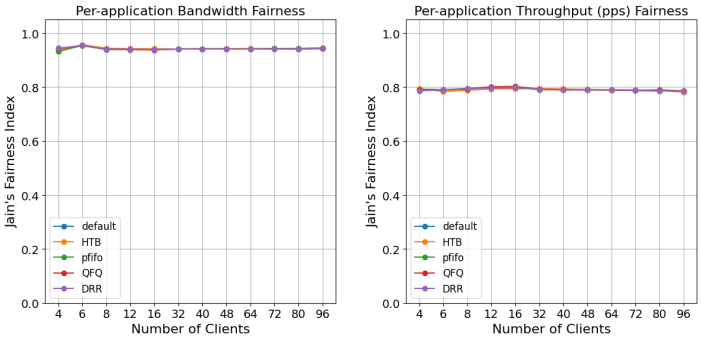
We begin the experiment by applying **S0**, where all flows use equal packet lengths, running concurrently. Using these ideal conditions, we hypothesize that the *qdiscs* will provide perfect fairness when using *Jain's Fairness Index* (equal to 1). We plot our data from section 5, using 3 different packet lengths, 1KB, 4KB and 64KB.

Using figure 6.7, we confirm our hypothesis; **the fairness index yields 1 (perfect fairness) for all *qdiscs*, when using 1KB (graph (a)) and 4KB (graph (b)) packet lengths (KF9)**. However, when the clients use a 64KB packet length (graph (c)), **around 5% of a drop is observed when using *HTB*, after 40 clients (KF10)**. We link this drop in fairness to the bandwidth degrading from section 5, with larger packet lengths. The same conditions also hold for fairness via per-application throughput (not displayed in the figures), where we compute the index via packets per second, which yields to the same exact plot as bandwidth fairness. Subsequently, while **S0** confirms our hypothesis on ideal conditions, it also highlights the need to test fairness using non-ideal and closer to realistic situations. This way, we motivate our choice for a second evaluation, **S1**.



(a) (b) (c)

Figure 6.7: Jain's Fairness Indexes calculated with per-application bandwidth for packet lengths of 1KB, 4KB and 64KB.



(a) (b)

Figure 6.10: Jain's Fairness Indexes calculated with per-application bandwidth and per-application throughput with a base packet length of 4KB

We continue with **S1**, where we introduce different packet lengths to observe the effect on fairness. Firstly, unlike the previous experiment, we discard 2 clients and start with 4 clients directly due to the use of 3 different packet lengths. For our introduction of varying conditions, we use a base packet length of 4KB (as motivated before) which bounds to 50% of the flows. The 25% of the flows get 1KB while the rest of the 25% gets 8KB packet lengths, imitating more realistic conditions when compared to **S0**. We chose the packet lengths to observe differences in fairness while not effecting system performance from other perspectives (such as CPU utilization).

In figure 6.10, we firstly observe that, when compared to the previous experiment **S0**, our varying packet lengths have caused a light drop in fairness of around 2-3% – when measured using per-application bandwidth on graph (a). However, **when we compute**

the index using per-application throughput (in packets per second), we observe a significant decrease in fairness of around 20% when compared to S0 (not displayed on the figures) (KF11). We reveal that, **even when bandwidth fairness suggests near ideal behavior, flows experience less-fair treatment in terms of packets per second (KF12)**, pointing out that smaller packets are disadvantaged.

Within **S1**, we also highlight the importance of inspecting fairness from two different perspectives, bandwidth and throughput. Lastly, even when we deviate from ideal conditions by using varying packet lengths, we observe the non-negligible drop in fairness.

6.4 Summary and Key Findings

We list the key findings within the experiments performed in section 6.2 and 6.3 as the following:

- **KF8**: We observe an improvement of 10% to 350% in latency for the higher priority class when using *HTB* with a strict prioritization scheme in our setup, when compared to no prioritization.
- **KF9:** In our setup, we observe a perfect fairness index, under ideal conditions with packet lengths of 1KB and 4KB.
- **KF10**: In our setup, we observe 5% drop in the fairness index of *HTB* after 40 clients.
- **KF11**: We find that packet fairness decreases by 20% when flows of different packet lengths are introduced while using our setup.
- **KF12**: In our setup, we find that, even when bandwidth fairness suggests near ideal behavior, flows experience less-fair treatment in terms of packets per second

Using our key findings, we answer **RQ3**, *How and to what extent does TC ensure the previously determined performance QoS properties in multi-tenant setups?* in two parts: **(1)** In terms of prioritization, within our setup, *HTB* reduces the latency of prioritized classes up to 350%. **(2)** In terms of fairness in our setup, all *qdiscs* perform identical, yielding a perfect fairness score under ideal conditions and a high-score (on average) of 0.95 when calculated with bandwidth and 0.8 when calculated via packets per second.

7

Reporting Negative Results and Limitations

7.1 Negative Results

Bandwidth drop on increasing clients has been observed throughout the experiments on **RQ2**, where we hypothesis an increase due to more flows sending more packets. While we tried to mitigate the observation by applying various micro-benchmarks, optimizations and system setting changes, we have not found a method to do so. We believe that the issue lies within either the underlying TCP or iperf3 algorithms, which stand out of our scope of investigation. Nevertheless, the certainty does not effect the results as the *qdiscs* are compared within the same environment, where the behavior of the system and TCP is the same.

7.2 Limitations

Limitations on the system setup concern generalizability, as our experiments are conducted using a single system setup, they leave uncertainty when applied within different systems with different characteristics – such as but not limited to different *Linux* versions, *NICs* or system configurations.

Limitations on the tools concern generalizability, especially when using benchmark generators and tools that replicate connections and packets. Tools such as *iperf3* and *netperf* do not represent real world scenarios where flows experience truly random and flawed behavior.

7. REPORTING NEGATIVE RESULTS AND LIMITATIONS

Limitations on prioritization rise as we could not evaluate the prioritization of all of the *qdiscs* due to them not supporting classful behavior or strict prioritization. Additionally, our experiments are limited to a single prioritization scenario, whereas more varied workloads (such as more prioritization classes) should be evaluated to amplify the findings.

Limitation on fairness include experiments being performed under close to ideal workloads, where flows are essentially indistinguishable. While we utilize varying packet lengths within a single experiment, our approach did not stand sufficient to fully evaluate fairness of *qdiscs* under realistic scenarios. Introducing factors such as delay or bursts to assess fairness will strengthen the experiments and the conclusions.

8

Related Work

In the field of networking, evaluating the performance of tools becomes a key element in designing, choosing and utilizing both conceptual and practical systems. Evaluating and optimizing certain tools and systems not only enables efficiency but also paves way for further research and development. Thus many others are exploring efficient and optimized frameworks and tools that aid development.

Evaluating the performance of tools within the *state-of-the-art* provides an understanding of the tools and further optimizations. As an example, in (56), *Cai et al.* measure Linux network stack performance in high-speed environments, exceeding 100Gbps, highlighting bottlenecks in kernel optimizations. Unlike our scope on evaluating specific QoS needs and overhead expectations, they focus on highlighting when optimizations are needed, thus recommending the use of certain optimizations – as an example, *TSO* – for specific cases. This allows other researchers and developers in using their recommendations to optimize and/or adapt their work to certain scenarios.

Implementing and evaluating tools becomes fundamental as the networking field scales. As an example, *Tu et al.* address the limitations of *Open vSwitch* in modern-day applications in (58) and present an optimized alternative. They highlight that such tools require modern day optimizations and revisions to keep up with the current advancements. On the other hand, in (43), *Alizadeh et al.* address the advancements by simplifying their approach. In the paper they present *pFabric*, in their words, a *"minimalistic datacenter transport design that provides near theoretically optimal flow"*. They provide an algorithm which can separate flow scheduling and rate control and show that simplicity plays an important role in optimizing flow performance. Additionally, they evaluate their implementation and explore the effect of different buffer sizes and rate control algorithms. Similarly,

Homa (32), a receiver based transport protocol, achieves improved message complete times when compared to *pFabric* – especially when evaluated with mixed workloads.

Advancements in other domains, require as much input as on Linux based datacenter networking. As an example, *Kumar et al.* in (59) evaluate user-space network stacks within a mobile 5G core, highlighting both the capabilities of an optimized stack and improving QoS within next-generation networks.

9

Conclusion

In this thesis, we have approached TC and QoS using a survey (section 3) and two evaluations (section 5 and 6) to answer our main **RQ**, *How does TC control performance QoS in the network and to what extent does it provide performance, scalability and QoS when applied to high speed and scalable networks?*

9.1 Answering Research Questions

RQ1 ***How do we define performance QoS in high-speed networks and how does the state-of-the-art use TC to provide these performance QoS properties?***

To address this question, we have conducted a survey, divided into two parts – TC and QoS –, aiming to explore *state-of-the-art* research, focusing on high-speed networks.

Firstly, we find that QoS definitions differ significantly among the context, thus requiring a carefully organized scheme when utilized for a specific cause. As for this, we focus on 3 properties – being *work-conservation*, *fair-sharing* and *prioritization* – where we define the properties and metrics to be used within our context – high-speed networks. We address the properties as follows **(1)** *work-conservation* as *per-application bandwidth* and *per-application throughput*, **(2)** *fair-sharing* using *Jain's Index* and lastly **(3)** *prioritization* as *average latency*.

Secondly, we explore papers that utilize TC as either a key-component within their design or a tool to complete their research. For this reason, we summarize and divide TC into 4 main titles, and observe how the *state-of-the-art* use TC to dispatch the concepts. In our survey, TC's use is summarized in **(1)** *Bandwidth Enforcement and Rate Limiting*, **(2)** *Traffic Shaping and Scheduling*, **(3)** *QoS Enforcement* and **(4)** *Network Emulation*. This

way we not only answer our research question which focuses on QoS properties, but we also provide a better understanding of TC as a tool.

Using our findings from the survey, we not only clarify QoS definitions and TC employment, but we also pave the groundwork of our experiments, which rely on **RQ1**.

RQ2 ***What is the performance overhead in bandwidth, CPU and latency of using TC?***

To answer the question, we have evaluated *qdiscs* under both increasing packet lengths and increasing clients, following our method in section 5.1.

We find that TC introduces a non-negligible overhead, when using *HTB*, which show high CPU utilization and bandwidth limitations in cases of contention. Using our setup, we have found between 5% to 30% CPU overhead with packet lengths greater than 1KB. On the other hand, while the difference in latency of the *qdiscs* are negligible when using 1KB and 4KB packet lengths, we observe an increase in latency of 20% (on average) when using *HTB*.

RQ3 ***To what extent does TC ensure the previously determined performance QoS properties in multi-tenant setups?***

To answer the question, we assess two of the QoS properties defined under section 2 while answering **RQ1** – prioritization and fairness.

We show that in terms of prioritization in our setup, *HTB* reduces the latency of classes with high priority by up to 350% under contention scenarios, proving it to be efficient for prioritization.

Secondly, we show that in our setup, all *qdiscs* achieve perfect fairness under ideal conditions. With varying packet lengths, *qdiscs* still show a consistent fairness among them, while *Jain's Fairness Index* yields a high score. However, once we calculate the index via packets instead of bandwidth, we observe a 20% drop, highlighting disadvantages of flows with smaller packet lengths.

Main RQ ***How does TC control performance QoS in the network and to what extent does it provide performance, scalability and QoS when applied to high***

speed and scalable networks?

Throughout our survey and our experimental approaches, we have found how TC controls and enforces performance and QoS goals such as bandwidth, latency, tenant prioritization and tenant fairness. However, its effectiveness, overhead and the extent it meets these performance objectives is shaped by the underlying *qdisc*, flow characteristics and the scale of deployment.

We find that TC's overhead in CPU and latency is tied to *qdisc* and load selection. Particularly, we observe that *HTB* introduces additional CPU overhead, especially in high contention under multi-tenant situations. Additionally, we reflect the effect of using greater packet lengths and its effect on *HTB*, where we observe an increase in latency and a decrease in bandwidth.

Additionally, our findings suggest that TC enforce the evaluated QoS behaviors (prioritization and fairness) when assessed using our setup, such as improved latency for prioritized flow classes and consistent fairness when scaled. These observations are obtained under specific controlled conditions, where we test prioritization only for *HTB* and evaluate both properties using a single environment, helping us to highlight the importance of context in interpreting QoS properties.

Overall, we believe that TC provides performance QoS in scalable networks, but its practicality relies on careful configuration of the *qdiscs* and workloads.

Using our findings and conclusions which answer the **RQ**s, we recommend the following for the use of TC and QoS definitions in the context of high speed, scalable data centers:

- We recommend defining performance QoS properties and metrics contextually, based on the system characteristics, workload selection and network goals.
- We recommend trade-off aware deployment and configuration of TC, as each *qdisc* highlights a trade-off between scalability, QoS enforcement and overhead.
- We recommend the selective deployment of *HTB*, due to its CPU overhead and performance decrease, especially in high-load environments, unless strict prioritization is required.

- We recommend the consideration of both per-byte (bandwidth) and per-packet (throughput) when assessing fairness with *Jain's Fairness Index* to ensure accurate fairness evaluation.

9.2 Future Work

Our research supports future investigation in different directions. To start with, further research on the effect of filters when classifying flows on overhead or the properties could provide deeper understanding of how classification effects the performance of the *qdiscs*, especially when under high contention due to increased number of clients.

Additionally, while we only focused on fairness and prioritization for the evaluation part, future studies could focus on extending the QoS definition and asses properties such as but not limited to work-conservation and energy efficiency. Further on, we leave the exploration of the QoS properties – prioritization and fairness – under distinct scenarios to future work.

Lastly, the thesis paves way for research regarding the broader exploration of TC and its *qdiscs* as modern network demands continue to evolve and grow. We highlight the need for constant evaluation and development of tools and mechanisms that are designed to balance performance, overhead and QoS properties in modern-day, scalable and high-speed data centers.

References

[1] Alexandru Iosup, Fernando Kuipers, Ana Lucia Varbanescu, Paola Grosso, Animesh Trivedi, Jan Rellermeyer, Lin Wang, Alexandru Uta, and Francesco Regazzoni. **Future Computer Systems and Networking Research in the Netherlands: A Manifesto**, 2022. 1

[2] Zili Meng, Tingfeng Wang, Yixin Shen, Bo Wang, Mingwei Xu, Rui Han, Honghao Liu, Venkat Arun, Hongxin Hu, and Xue Wei. **Enabling high quality {Real-Time} communications with adaptive {Frame-Rate}**. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1429–1450, 2023. 1

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. **A view of cloud computing**. *Commun. ACM*, **53**(4):50–58, April 2010. 1, 14

[4] Stefanos Antaris, Dimitrios Rafailidis, and Sarunas Gidzijauskas. **A deep graph reinforcement learning model for improving user experience in live video streaming**. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1787–1796. IEEE, 2021. 1

[5] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. **A large scale study of data center network reliability**. In *Proceedings of the Internet Measurement Conference 2018*, pages 393–407, 2018. 1

[6] Amazon Web Services. **Amazon Web Services (AWS) - Cloud Computing Services**, 2025. 1

[7] IBM. **IBM Cloud | Cloud Computing Services**, 2025. 1

[8] Google. **Google Data Centers**, 2025. 1

[9] Zhi-Wei Xu. **Cloud-Sea Computing Systems: Towards Thousand-Fold Improvement in Performance per Watt for the Coming Zettabyte Era**. *Journal of Computer Science and Technology*, **29**(2):177–181, Mar 2014. 1

[10] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. **Data center TCP (DCTCP)**. *SIGCOMM Comput. Commun. Rev.*, **40**(4):63–74, August 2010. 1

[11] International Data Corporation (IDC). **Worldwide Cloud IT Infrastructure Market Share, 2024**. `https://my.idc.com/getdoc.jsp?containerId=US50267124`, 2024. Accessed April 18, 2025. 2

[12] Jamal Hadi Salim, Robert Olsson, and Alexey Kuznetsov. **Beyond softnet**. In *5th Annual Linux Showcase & Conference (ALS 01)*, 2001. 2

[13] Jiaxin Lin, Adney Cardoza, Tarannum Khan, Yeonju Ro, Brent E Stephens, Hassan Wassel, and Aditya Akella. {**RingLeader**}**: efficiently Offloading** {**Intra-Server**} **Orchestration to** {**NICs**}. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1293–1308, 2023. 2

[14] National Laboratory for Applied Network Research (NLANR). **National Laboratory for Applied Network Research**, 2025. Accessed April 18, 2025. 2

[15] Hewlett Packard. **Netperf: A Network Performance Benchmark**, 2025. Accessed April 18, 2025. 2

[16] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, et al. **Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches**. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 194–206, 2021. 7

[17] Alibaba. **Alibaba Cloud**, 2025. 7

[18] Zhiyu Zhang, Shili Chen, Ruyi Yao, Ruoshi Sun, Hao Mei, Hao Wang, Zixuan Chen, Gaojian Fang, Yibo Fan, Wanxin Shi, et al. **vPIFO: Virtualized**

Packet Scheduler for Programmable Hierarchical Scheduling in High-Speed Networks. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 983–999, 2024. 7

[19] Miquel Moreto, Francisco J Cazorla, Alex Ramirez, Rizos Sakellariou, and Mateo Valero. **FlexDCP: a QoS framework for CMP architectures**. *ACM SIGOPS Operating Systems Review*, **43**(2):86–96, 2009. 14

[20] Robert Braden, David Clark, and Scott Shenker. **Integrated services in the internet architecture: an overview**. 1994. 14

[21] Steven Blake, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss. **Rfc2475: An architecture for differentiated service**, 1998. 14

[22] Inayat Ali, Seungwoo Hong, and Taesik Cheung. **Quality of Service and Congestion Control in Software-Defined Networking Using Policy-Based Routing**. *Applied Sciences*, **14**(19), 2024. 14

[23] Zhao Weibin, D Olshefski, and H Schulzrinne. **Internet quality of service: An overview**. *Technical Report CUCS-003-00*, 2000. 14

[24] Awada Uchechukwu, Keqiu Li, Yanming Shen, et al. **Energy consumption in cloud computing data centers**. *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, **3**(3):31–48, 2014. 14

[25] John Judge, Jack Pouchet, Anand Ekbote, and Sachin Dixit. **Reducing data center energy consumption**. *Ashrae Journal*, **50**(11):14, 2008. 15

[26] Huigui Rong, Haomin Zhang, Sheng Xiao, Canbing Li, and Chunhua Hu. **Optimizing energy consumption for data centers**. *Renewable and Sustainable Energy Reviews*, **58**:674–691, 2016. 15

[27] Ralph Hintemann and Simon Hinterholzer. **Energy consumption of data centers worldwide**. *Business, Computer Science (ICT4S)*, 2019. 15

[28] Md Abu Bakar Siddik, Arman Shehabi, and Landon Marston. **The environmental footprint of data centers in the United States**. *Environmental Research Letters*, **16**(6):064017, may 2021. 15

[29] Lucian Popa, Praveen Yalagandula, Sujata Banerjee, Jeffrey C. Mogul, Yoshio Turner, and Jose Renato Santos. **ElasticSwitch: practical work-conserving bandwidth guarantees for cloud computing**. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, page 351–362, New York, NY, USA, 2013. Association for Computing Machinery. 15, 17

[30] Shuihai Hu, Wei Bai, Kai Chen, Chen Tian, Ying Zhang, and Haitao Wu. **Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud**. *IEEE Transactions on Cloud Computing*, **9**(2):763–776, 2018. 15

[31] Zhuotao Liu, Kai Chen, Haitao Wu, Shuihai Hu, Yih-Chun Hut, Yi Wang, and Gong Zhang. **Enabling work-conserving bandwidth guarantees for multi-tenant datacenters via dynamic tenant-queue binding**. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2018. 15

[32] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. **Homa: A receiver-driven low-latency transport protocol using network priorities**. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 221–235, 2018. 15, 44

[33] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. **Less is more: Trading a little bandwidth for {Ultra-Low} latency in the data center**. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 253–266, 2012. 16

[34] Huaizhou SHI, R. Venkatesha Prasad, Ertan Onur, and I.G.M.M. Niemegeers. **Fairness in Wireless Networks:Issues, Measures and Challenges**. *IEEE Communications Surveys Tutorials*, **16**(1):5–24, 2014. 16

[35] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. **A quantitative measure of fairness and discrimination**. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA*, **21**(1), 1984. 16

[36] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari

Sridharan. **Data center tcp (dctcp)**. In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010. 17

[37] Haoyu Wang, Xiaoqian Zhang, Allen Yang, and Bo Sheng. **DCTCP-FQ: Enhancing Fairness and Convergence Time in Data Center Congestion Control**. In *2025 International Conference on Computing, Networking and Communications (ICNC)*, pages 699–703. IEEE, 2025. 17

[38] Pulkit A Misra, María F Borge, Íñigo Goiri, Alvin R Lebeck, Willy Zwaenepoel, and Ricardo Bianchini. **Managing tail latency in datacenter-scale file systems under production constraints**. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–15, 2019. 17

[39] Jialin Li, Naveen Kr Sharma, Dan RK Ports, and Steven D Gribble. **Tales of the tail: Hardware, os, and application-level sources of tail latency**. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–14, 2014. 17

[40] Gaoxiong Zeng, Li Chen, Bairen Yi, and Kai Chen. **Optimizing tail latency in commodity datacenters using forward error correction**. *CoRR*, 2021. 17

[41] Jeffrey Dean and Luiz André Barroso. **The tail at scale**. *Communications of the ACM*, **56**(2):74–80, 2013. 17

[42] Kevin Zhao, Prateesh Goyal, Mohammad Alizadeh, and Thomas E. Anderson. **Scalable Tail Latency Estimation for Data Center Networks**. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 685–702, Boston, MA, April 2023. USENIX Association. 17

[43] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. **pFabric: Minimal near-optimal datacenter transport**. *ACM SIGCOMM Computer Communication Review*, **43**(4):435–446, 2013. 17, 43

[44] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. **PIAS: Practical information-agnostic flow scheduling for commodity data centers**. *IEEE/ACM Transactions on Networking*, **25**(4):1954–1967, 2017. 18, 21

[45] Matthew P Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert NM Watson, Andrew W Moore, Steven Hand, and Jon Crowcroft. **Queues**

{**don't**} **matter when you can {JUMP} them!** In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 1–14, 2015. 18, 20

[46] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. **Hopper: Decentralized Speculation-aware Cluster Scheduling at Scale**. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 379–392, New York, NY, USA, 2015. Association for Computing Machinery. 18

[47] Fatima Tul Zuhra, Kamalrulnizam Bin Abu Bakar, Adnan Ahmed Arain, Khaled Mohamad Almustafa, Tanzila Saba, Khalid Haseeb, and Naveed Islam. **LLTP-QoS: low latency traffic prioritization and QoS-aware routing in wireless body sensor networks**. *IEEE Access*, **7**:152777–152787, 2019. 18

[48] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, et al. **BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing**. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 1–14, 2015. 19

[49] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. **Guaranteeing deadlines for inter-datacenter transfers**. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–14, 2015. 19

[50] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. **Carousel: Scalable traffic shaping at end hosts**. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 404–417, 2017. 20

[51] Peter P Waskiewicz Jr. **Converged Networking in the Data Center**. In *Linux Symposium*, page 297. Citeseer, 2009. 20

[52] Junxue Zhang, Wei Bai, and Kai Chen. **Enabling ECN for datacenter networks with RTT variations**. In *Proceedings of the 15th international conference on emerging networking experiments and technologies*, pages 233–245, 2019. 21

[53] Soeren Becker, Tobias Pfandzelter, Nils Japke, David Bermbach, and Odej Kao. **Network emulation in large-scale virtual edge testbeds: A note of caution and the way forward**. In *2022 IEEE International Conference on Cloud Engineering (IC2E)*, pages 1–7. IEEE, 2022. 21

[54] Tariq Emad Ali, Ameer Hussein Morad, and Mohammed A Abdala. **Traffic management inside software-defined data centre networking**. *Bulletin of Electrical Engineering and Informatics*, **9**(5):2045–2054, 2020. 21

[55] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. **The design and implementation of open {vSwitch}**. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, pages 117–130, 2015. 21

[56] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. **Understanding host network stack overheads**. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 65–77, New York, NY, USA, 2021. Association for Computing Machinery. 24, 43

[57] Erfan Sharafzadeh, Raymond Matson, Jean Tourrilhes, Puneet Sharma, and Soudeh Ghorbani. **{Self-Clocked}{Round-Robin} Packet Scheduling**. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 1437–1465, 2025. 24

[58] William Tu, Yi-Hung Wei, Gianni Antichi, and Ben Pfaff. **Revisiting the open vswitch dataplane ten years later**. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 245–257, 2021. 43

[59] Ashwin Kumar, Priyanka Naik, Sahil Patki, Pranav Chaudhary, and Mythili Vutukuru. **Evaluating network stacks for the virtualized mobile packet core**. In *Proceedings of the 5th Asia-Pacific Workshop on Networking*, pages 72–79, 2021. 44

Appendix A

Reproducibility

A.1 Abstract

The artifact includes the benchmarking framework used to evaluate *qdiscs* on sections 5 and 6. It includes the scripts as well as the obtained raw data and the plots from the experiments, for different packet lengths, number of clients and varying conditions. The artifact ensures reproducibility of the results commented on the previous sections, including the setup and configurations used.

A.2 Artifact check-list (meta-information)

- **Program:** TC-benchmark (`https://github.com/yyyeeettt3/TC-Benchmark`)
- **How much time is needed to prepare workflow (approximately)?:** 15 minutes
- **How much time is needed to complete experiments (approximately)?:** between 2 and 11 hours (depending on the experiments)
- **Publicly available?:** The benchmarking framework is publicly available via the *GitHub* repository
- **Code licenses (if publicly available)?:** MIT License

A.3 Description

A.3.1 How to access

All artifacts (the configurations, scripts and data) are available in the public *GitHub* repository (`https://github.com/yyyeeettt3/TC-Benchmark`)

A.3.2 Hardware dependencies

The hardware is expected to have multiple cores (more than 4) and a high-speed NIC (our experiments reach 50Gbit/s bandwidth).

A.3.3 Software dependencies

The benchmarks are designed to be run on the following (or compatible) dependencies:

- Ubuntu Linux 20.04+
- Kernel version 5.12.01
- *iperf3* version 2.10.0
- *netperf* version 2.6.0

A.4 Experiment workflow and Customization

Experiments can be found under `/experiments`, making our evaluations in section 5 and 6 replicable. For any script, modifying the `QDISCS` and `QDISCS_SETUP` within the files allows the addition (or removal) of *qdiscs* (additional setup files are needed for *qdiscs* which are note evaluated previously). Additionally, the scripts are modifiable in terms of packet length, number of clients, number of repetitions and tool configurations, which can all be modified via the script files, as described in `README.md`.

The scripts can be executed as `./name-of-the-script`. After executing the scripts the files, results are saved under `/replicated_results`, whereas our obtained results are under `results`. Additionally, we include our plotting scripts saved under `/processing`.

A.5 Evaluation and expected results

The results when our experiments are replicated should be similar to our results in section 5 and 6. However, different setups, both in terms of hardware or software could impact the results.