Interplay Between Priority Queues and Controlled Delay in Programmable Data Planes

Tung V. Doan*, Tobias Scheinert†, Osel Lhamo†, Juan A. Cabrera†,
Frank H. P. Fitzek†‡, Giang T. Nguyen*‡
*Haptic Communicatioon Systems, TU Dresden, Germany
†Deutsche Telekom Chair of Communication Networks, TU Dresden, Germany
‡Centre for Tactile Internet with Human-in-the-Loop (CeTI), TU Dresden
E-mails: {firstname.lastname}@tu-dresden.de, tobias.scheinert@mailbox.tu-dresden.de

***Abstract*—The Tactile Internet needs to support multiple modalities, including haptic data simultaneously, to enable human and robot interaction in perceived real-time. Compared to video and audio, haptic data streams require an even lower delay and more shallow packet loss, which is a challenge for the underlying networks originally designed to support high bandwidths by having a large queue to buffer packets. This raises the need to study queuing management for different data streams. We examine the performance of CoDel++, which combines the well-known Controlled Delay (CoDel) protocol and Priority Queues. We leverage programmable data planes, which can be programmed with P4. Evaluation results show that CoDel++ produces superior results for high-priority data streams, i.e., reducing latency by at least 60% and packet loss by half compared to individual exploitation of Priority Queuing or CoDel individually.**

***Index Terms*—Tactile Internet, Buffer Bloat, Active Queue Management, Congestion Control, P4**

I. Introduction

The Internet has been successfully connecting people across the globe by delivering many types of data, including audio and video. Therefore, the Internet has enabled multimedia applications, such as audio and video streaming and real-time video conferences. Furthermore, novel applications, such as teleoperation, allow humans to remotely control robots from a distance, leveraging audio and video streams as supporting means. Human adaptability and advanced robot-controlling algorithm help overcome the discrepancy of interactions due to latency. However, perceived real-time interactions require the sense of touch via haptic data streams [1]. It is, however, very challenging because human brains are significantly more sensitive to touch stimuli than acoustic and visual ones. An authentic feel during the interaction would require the delivery of haptic data streams with low latency and extremely low packet loss. Enabling such interaction via better support of haptic data streams is the goal of the future Tactile Internet [2]. Such networks can leverage novel yet mature technologies, such as software-defined networking and programmable data planes, to tackle fundamental challenges in computer networking. One of which is buffer bloat.

Buffering packets at network devices is a standard practice in networking. The goal is to avoid packet drops at the cost of increasing dissemination delay. The problem becomes extreme when buffers have a high occupation ratio for extended periods (so-called buffer bloat), increasing the overall delay and eventually undermining the buffering role. To tackle the buffer bloat problem, Active Queueing Management is a well-known technique that discards packets in a buffer connected to a network interface controller before this buffer is full, often to reduce network congestion or improve latency. For example, the Random Early Detection (RED) algorithm [3] alleviates congestion by keeping track of the average queue length and dropping packets prematurely as the buffer nears capacity. The drawback of RED is centrally on its parameters. Fine-tuning various parameters is difficult and time-consuming. Furthermore, RED's complexity can lead to performance degradation. In a different approach, Controlled Delay (CoDel) [4] aims to resolve the parameter setting problem and combat the constant buffer bloat issue. Specifically, CoDel operates like a FIFO queue where packets are stamped with an entrance time and sent to the device's statistical multiplexer upon entering the switch. The packets are handled in the order of their arrival [5]. Nevertheless, CoDel applies the same policies for all data flows, regardless of their differences in delay and packet loss requirements.

Motivated by this observation, in this study, we focus on understanding the impacts of CoDel++, combining priority queuing and CoDel to data flows of different priorities. We leverage the programmable data plane to design packet processing pipelines, which the P4 language can program. CoDel++ maps data flows, or slices, into priority levels, leverages the priority queues at the Traffic Manager, and integrates CoDel at the Egress. CoDel++ constantly observes the queue status and uses the corresponding readings to actively drop packets before they enter the queues based on pre-defined thresholds. We examine the performance of CoDel++ with CoDel and Priority Queuing individually and the baseline without any of those techniques. The performance evaluation on bottleneck topology suggests that the combination help reduce both delay and packet losses in congestion phases.

The rest of the paper is structured as follows: Section II discusses the background and related work. We elaborate on the methodology in Section III. After presenting the evaluation results in Section IV, we conclude the paper in Section V.

 ISBN: 978-3-903176-56-0

II. Related work

Congestion control is crucial in computer networks. As a result of the discontinuous nature of Internet traffic, congestion affects the network edge and core. AQM is performed by the network scheduler, which uses various algorithms [6] to help with the congestion problem. However, there were initial reservations about the widespread adoption of AQM policies due to stability issues. Nevertheless, over time operators worldwide have started applying AQM algorithms upstream of the ADSL modem to enhance the user's quality of experience [7].

Some of the potential treatments for congestion that helped build our paper will be covered in this section. The tail drop is, by default, used as a queue management algorithm to control congestion [8]. It is based on a first-in-first-out (FIFO) queue that drops arriving packets when the buffer is full. However, the issue with this algorithm is that the sender starts regulating its transmission rate only after noticing packet loss (resulting from buffer overflow). Detecting packet loss can sometimes take a significant amount of time, and by then, more packets are dropped due to the sender's high transmission rates.

To curb the disadvantages of the tail drop algorithm, Floyd and Jacobson introduced Random Early Detection (RED) algorithm [3], which operates by using two thresholds: a minimum threshold (min_{th}) and maximum threshold (max_{th}). No packet is discarded if $aql < min_{th}$ while all packets after $aql > max_{th}$ are dropped. If $min_{th} < aql < max_{th}$, then packets are dropped with a probability depending linearly on the buffer occupancy [9]. As already mentioned in the previous section, the limitations of RED are configuring several parameters for which little guidance is provided and its dependence on aql that can cause performance issues during sudden congestion. Even though there have been many modifications made to the original RED algorithm [10], [11], [12], these different flavors additionally complicate the process of RED, thus there continues to be much resistance to its widespread deployment [13].

Considering the shortcomings of previous AQMs, Nichols and Jacobson proposed Controlled Delay (CoDel) [4]. Compared to earlier AQMs, CoDel does not employ queue length as a parameter and instead uses queue delay as an indicator of congestion. According to [14], CoDel was created to be a parameterless, simple-to-implement AQM capable of differentiating" good" and" bad" queues, and is insensitive to round-trip delays, link rates, and traffic loads. CoDel regulates delay solely within the context of packet-in, packet-out. In addition, a set point and a state-space controller are utilized to preserve the algorithm's state. During the dequeue process, the state-space controller must assess whether a queue is bad or good by using the sojourn time and the set point. Sojourn time is a compelling statistic for traffic congestion. It is calculated as the difference between the packet's enqueue and dequeue timestamp. As congestion grows, this delay per packet will naturally increase. By utilizing an estimator function, CoDel enters a dropping state when the observed sojourn duration exceeds the previously indicated goal set point. CoDel has achieved great success. It was even introduced in the Linux kernel in 2012. However, its limitation lies in not enforcing different and specific rules depending on the QoS needs of particular applications. Thus, [5] describes an algorithm that combines CoDel with priority queuing. It assigns and evaluates priority classes and drops packets based on the Least-Slack-Time-First scheduling algorithm.

Many researchers even use SDN to resolve the congestion problem. Through a logically centralized controller, SDN enables simplified access and management of a network's control plane. OpenFlow is one of the first widely used protocols for enabling SDN. However, it has downsides, such as the inability of Internet service providers to select particular services, even if not all functions are always required. Additionally, costs are high, and new OpenFlow versions come up with different hardware. Considering data plane functionality is not predetermined by hardware, P4 was developed as an extension of the OpenFlow protocol that provides higher flexibility and a possibility to directly influence packet processing [15]. Nevertheless, unlike protocols such as Openflow that already have a variety of AQM algorithms at their disposal, this is not generally true for AQM on programmable data plane [16].

Although OpenFlow can already draw on a set of implemented congestion control algorithms, these must first be implemented on a corresponding P4 target architecture. Implementing a new proposed algorithm using P4 and the subsequent evaluation is part of this work. Similar to table slicing of [17], our P4 code divides flows into network slices (each owning a queue). While authors of [17] investigate different network slicing techniques and their implementation on P4 target, the focus of our work is to explore AQM algorithms (enabled with data plane programming) by using network slicing for different flows and schedule them according to their priority class. Priority scheduling is a common way to differentiate traffic flow inside a network by assigning a priority class to the packets as it was shown in [18], [5], and [19].

III. Designing Packet Processing Pipeline

This work studies the combined impacts of integrating priority queuing and CoDel in reducing both packet losses and delays for high-priority data streams. We leverage programmable data planes, which allow reprogramming the packet processing pipeline using the P4 programming language. We elaborate the whole pipeline for packet processing, as illustrated in Fig. 1, including modifications to incorporate priority queuing and controlled delay protocols. Since the Traffic Manager (TM) is nonadjustable due to the current development of P4, we keep TM unchanged. The main modification occurs in the Ingress and Egress modules, where we define actions to incoming packets. We integrate the priority queuing discipline and CoDel into the Ingress and Egress. In addition, we exploit registers to convey congestion status at the exit of the traffic manager in order to filter and drop packets already in the Ingress section. We modify the Parser and Deparser modules to define additional fields for packet headers.

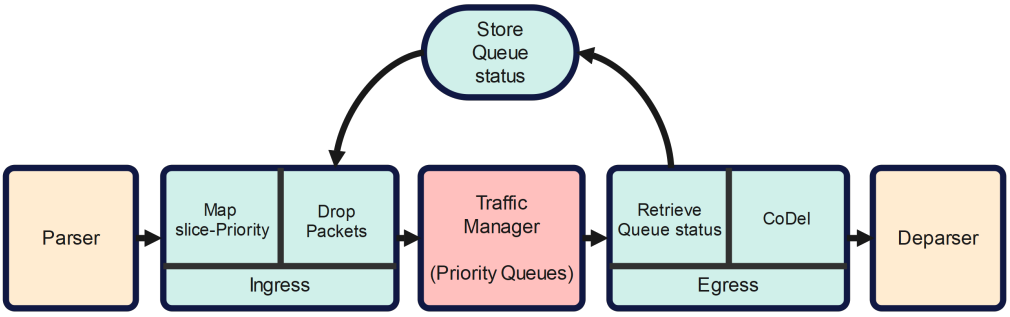


Fig. 1. Packet processing pipeline for CoDel++, integrating Priority Queuing and CoDel.

A. Defining Headers and Metadata

This section defines the header and metadata of packets to be processed by the programmable data plane, which includes any program-specific metadata associated with the processed packet and the format of each header within a packet. For example, listing 1 shows the declaration of standard packet headers with Ethernet, IPv4, and UDP or TCP that the P4 switches can recognize in this work. Additionally, each packet is associated with metadata. This user-defined data elaborates intermediate information generated during the execution of a P4 program. P4 programs can also retain and change metadata, as defined in the second part of listing III-A. CoDel user-defined metadata are inspected and taken from the implementation by Kundel et al. [20] as well as headers. In addition to other metadata declarations, the slice ID field has been adopted from [18]. Metadata for priority queuing contain prepared bit fields, which can be filled with extra information on congestion control in each packet. For every slice within the P4 switch, congestion notifications and queuing metrics are given and stored in user-defined registers. Their values can subsequently be connected with each packet via the metadata specified by the user.

```
struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
    tcp_t        tcp;
    udp_t        udp; }

struct metadata {
    codel_t      codel;
    prio_t       prio;
    slice_t      slice; }
```

Listing 1. Define headers and additional metadata.

The parser specifies the allowed header sequences within received packets. It recognizes and extracts specific header sequences from packets. The parser operates similarly to a state transition diagram, beginning with the start states and branching to other states, whose definitions are illustrated in Listing 2. The parser first extracts the packet's Ethernet header in the start state. If $ethetype == 0x800$, the parser switches to the state parse_ipv4. The packet's appropriate header is extracted as an outcome of traversing each stage. The values stored in these memory structures are then accessible to other pipelines [21].

In the subsequent *parse_ipv4* state, the parser covers the packet header extraction for the transport protocols. It extracts the 20-byte field in the IPv4 header and examines the Protocol field. The parser switches states to either *parse_tcp* or *parse_udp* depending on the protocol equal $8w6$ or $8w17$, respectively. If the protocol field matches either of the values, then the parser switches to the accepted state, marking the completion of extracting packet headers. Subsequently, the extracted packet headers are used for pre-defined procedures within the Ingress and Egress pipeline.

```
state start {
   packet.extract(hdr.ethernet);
   transition select(hdr.ethernet.ethertype) {
      16w0x800: parse_ipv4;
      default: accept; } }

state parse_ipv4 {
   packet.extract(hdr.ipv4);
   transition select(hdr.ipv4.protocol) {
      8w17: parse_udp;
      8w6: parse_tcp;
      default: accept; } }

state parse_tcp {
   packet.extract(hdr.tcp);
   transition accept; }

state parse_udp {
   packet.extract(hdr.udp);
   transition accept; }
```

Listing 2. Extraction headers for layers 2-4.

B. Integrating Priority Queues and CoDel

We decided to integrate priority queues at the Ingress stage so that actions like dropping packets can be taken early enough before they enter the queue inside the Traffic Manager. The Ingress pipeline, as part of the V1Model switch architecture, can be split into two principal parts. The first Ingress pipeline defines three match-action tables with the associated actions. Table 1 maps packets to $slice\_id$. As a

packet's IP source address matches a predefined value, the action *slicing* assigns the packet to a particular $slice\_id$. The module assigns a priority class to the packet based on the $slice\_id$. Table 2 maps $slice\_id$ to $prioirty\_id$. The match/action table *slice_priority* prepares the mapping of packets into the corresponding priority queue within the traffic manager. The action *set_priority* fixates the priority field in metadata to a predefined value [19]. While table 1 and table 2 target specifically network slicing, table 3 is a standard one. The table maps packets to the $egress\_port$ of the switch. The table is responsible for matching incoming packets to the correct output port of the switch based on the slice ID and ingress port. All three tables together enable a logical splitting of various data streams within a network consisting of P4 switches. The second ingress pipeline filters packets according to their slice ID and the associated priority.

Inspired by [22], we integrate CoDel at the Egress because, in this later stage, CoDel can still consider internal delays caused by the Ingress and the Traffic Manager as inputs for its algorithm. The single input that CoDel needs is the internal delay, which could be retrieved from the Traffic Manager. The internal delay is the time difference between when the packet enters the parser and when the packet leaves the Traffic Manager. Additionally, the Egress also collects queue depths of all the priority queues inside the Traffic Manager.

C. Coordinating Priority Queues and CoDel

The second part of the Ingress pipeline is named Egress feedback, which is responsible for receiving and assigning congestion control notifications from the Egress pipeline back to the Ingress module. We leverage the switch's register to store the Egress module stores these values and metrics as user-defined metadata. Even though the original CoDel performs probing to estimate link latency, we simplified this process by assuming a uniform delay of packets. This allows for avoiding the complexity of the estimation process and focusing on the interaction between priority queuing disciplines. During congestion in the network, we assume the same delays and queue length occupancy within the traffic manager are present for all incoming packets at a given time t_0. For this reason, the register values at time t_0 can be assigned to multiple packets simultaneously. The Ingress applies a dropping policy for packets belonging to a specific slice ID by comparing the actual queue length and the corresponding threshold of that queue length. It means one can achieve differentiated services for various slices based on queue length for packets in the traffic manager.

As packets pass through the traffic manager's queues, the Ingress updates the packet's delay and queue length into the packet's metadata. The Egress uses the metadata to estimate the current state of the switch starting already at the Ingress, leveraging registers and Ingress tables. During the congestion phase, the Ingress drops several packets according to their priority class and perceived queue length, and the Egress processes the remaining processed according to the CoDel AQM algorithm elaborated in section II. While pure priority queuing works only with fixed queue lengths, the combination of priority queuing and CoDel allows for dynamically adjusting priority classes' queue lengths, enabling an immediate reduction of queues by dropping packets [14]. While pure priority queuing assigns packets to queues according to their priority class, priority queueing with congestion awareness can directly control the maximum queue threshold from which packets are dropped off the priority classes. Both algorithms, priority queueing with congestion awareness and CoDel AQM, enable an immediate reduction of queues by dropping packets [14].

IV. Performance Evaluation

In this section, we seek the answer to how well our proposed queuing management scheme enables isolation between slices compared to state-of-the-art schemes. Toward this aim, we evaluate the proposed scheme using P4 behavioral model version 2 (bmv2), a reference P4 software switch that is widely adopted for P4 prototype implementation. We then introduce performance metrics, including latency, packet loss, and throughput, to assess the performance of the studied schemes. Afterward, we detail our measurement setup and discuss the obtained results.

A. Metrics

To investigate the performance of the studied schemes, we consider three key performance metrics: latency, packet loss, and throughput. These performance metrics are widely adopted to evaluate queuing management schemes [22], [18], [23], [24], [25]. Specifically, a queuing management scheme is assessed by its ability in terms of network congestion. Due to the network congestion, network devices have to buffer packets, thus increasing packet latency (i.e., queuing delay) and reducing throughput. Moreover, network devices might need to drop packets to alleviate the congestion, consequently leading to packet loss. In this study, we used iPerf [26] to measure latency, packet loss, and throughput.

1) Latency: Even though Round-trip-time (RTT) and one-way-delay (OWD) are widely used in the literature to measure latency, we adopted OWD, that only calculates the time taken to transfer a packet from sender to receiver to provide a reliable latency measurement. More specifically, we used iPerf in UDP mode to measure the OWD. As both sender and receiver run on the same machine, the OWD measurement does not require clock synchronization between sender and receiver. The OWD typically includes transmission delay, link propagation delay, queuing delay, and processing delay. Considering the congestion scenario in an emulation environment (i.e., Mininet), the queuing delay dominates the OWD.

2) Packet loss: Packet loss represents lost packets that do not reach the receiver when transferred over the network. For queuing management, packet loss occurs when network devices drop packets in the queues to alleviate network congestion. In our study, the processing pipelines actively remove packets by invoking the mark_to_drop() function when a defined condition is met.

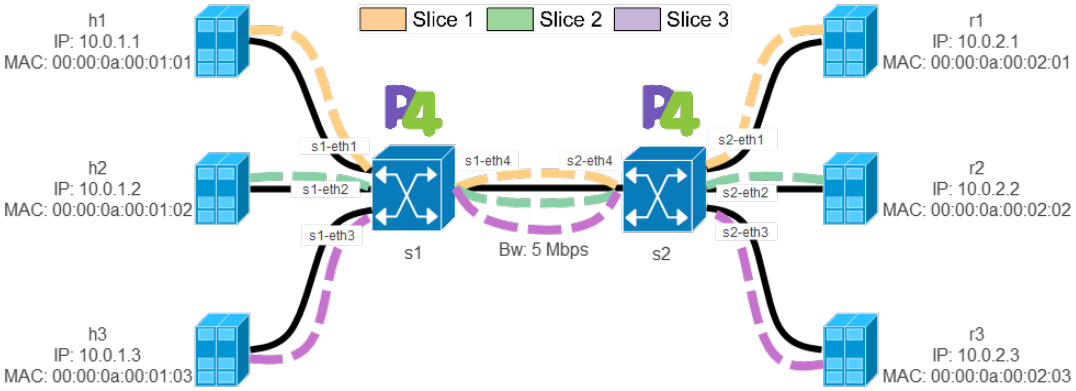


Fig. 2. Evaluation Topology with 3 senders and 3 receiver over a bottleneck link.

3) Network throughput: Network throughput measures the amount of data transferred from sender to receiver within a given time period. Queuing management schemes decide how long packets stay in the queues and thus directly affect network throughput.

B. Measurement Setup

Our evaluation attempts to understand the abilities of different queuing management schemes to differentiate services. Toward this end, we conduct a measurement setup to assess the performance of the studied queuing management schemes. Specifically, we first describe network topology, followed by the setting of different slices. Afterward, we detail our emulation setup that is used to build the network topology. We then present the settings for input and output data. Finally, we describe four benchmark schemes: Baseline, Priority Queuing, CoDel, and CoDel++.

To investigate the performance of the studied queuing management schemes in network congestion, we consider a bottleneck network topology [6]. Fig. 2 illustrates our network topology, which consists of six emulated hosts and two P4 software switches. In Fig. 2, three hosts on the left (h1, h2, and h3) are senders. Meanwhile, the other three hosts on the right (r1, r2, and r3) are receivers. The senders are connected to the receivers via two P4 software switches (s1 and s2). Specifically, the senders are linked to s1, while the receivers are linked to s2. The senders reach the receivers via the link between s1 and s2.

We proceed to present the setting of different slices. Toward this end, we set up three network slices (slice 1, slice 2, and slice 3) sharing the same underlying network mentioned above. Specifically, slice 1, slice 2, and slice 3 carry traffic from h1 to r1, h2 to r2, and h3 to r3, respectively. The P4 switches s1 and s2 differentiate the slices by assigning an ID to each slice in match/action tables to indicate the processing priority. Accordingly, slice 1 has the highest processing priority, while slice 3 has the lowest.

For the emulation setup, we consider an emulator that can rapidly prototype our network topology. Mininet [27] is a well-known tool that allows one to quickly emulate hosts, links, and switches on a single machine. However, Mininet does not support the P4 software switches for implementing queuing management schemes. Consequently, we adopted P4-Utils [28], an extension to Mininet that relies on the behavioral model (bmv2) [29] to provide the P4 software switches. Moreover, P4-Utils uses a reference compiler [30] to compile programs deployed on the P4 software switches. To provide isolation between slices, we installed forwarding rules on the switches to ensure that only hosts in the same slice can communicate with each other. We set the maximum link bandwidth to mimic the network congestion to 5 Mbps. We set the queue rate (i.e., the number of packets processed by the queue in one second) to 500 PPS based on the maximum link capacity.

We then describe the input data generated for different slices and the output data obtained from the measurement. For the input data, we used the iPerf client on senders to generate three data streams (UDP traffic) for slice 1, slice 2, and slice 3, respectively. The bit rates used for the data streams of slice 1, slice 2, and slice 3 are, in turn, 2 Mbps, 1 Mbps, and 3 Mbps. The sum of the bit rates for all data streams (6 Mbps) exceeds the link capacity, thus leading to the queues formed in the switches. To obtain the results, we used the iPerf server on receivers to receive and analyze the traffic from the senders. The experimental results represent the averages and 95% confidence intervals of 100 measurements, whereby each measurement ran in 10 seconds.

We proceed to describe four scenarios that we carried out to observe their performance for different slices:

- **Baseline**: This scheme uses default First-In-Fist-Out (FIFO) queues in the P4 switches s1 and s2. Specifically, the traffic manager in s1 and s2 sorts all incoming packets according to their arrival and buffers them if the queues form at the output port. Each FIFO queue can hold up to 1000 packets by default.
- **Priority Queuing**: We configured the P4 switches s1 and s2, similar to the baseline scheme. Different priorities

have been assigned to the slices. In particular, slice 1 has the highest priority, while slice 3 has the lowest priority.

- **CoDel**: We implemented CoDel [14] congestion control algorithm in the P4 switches s1 and s2.
- **CoDel++**: We proposed and implemented a new queuing management scheme that combines the priority queuing scheme and CoDel.

Note that for CoDel and CoDel++, we fixed the number of packets a queue can hold for each output port by setting the maximum queue depth to 10000, similar to [22]. This setting is larger than required to ensure that packet dropping is not caused by the full buffers.

C. Results

1) Latency: We first investigate the performance of the studied schemes in terms of latency. Fig. 3 shows the OWD measurement results of the studied schemes for different slices. Notably, Fig. 3 shows that CoDel++ performs the best for slice 1 among the studied schemes.

As CoDel and CoDel++ are two schemes that rely on a congestion control algorithm, we observe from Fig. 3 that CoDel and CoDel++ achieve better performance and obtain more stable results than the baseline scheme. Specifically, the OWD of CoDel for each slice is 448 ms. CoDel reduces the OWD by 70% for each slice compared to the baseline scheme. This superiority is because CoDel rapidly drops packets in the queues to alleviate the congestion. Meanwhile, CoDel++ is the combination of CoDel and the priority queuing scheme. Since CoDel++ assigns the highest priority to slice 1, the OWD of CoDel++ for slice 1 is less than for slice 2 and slice 3. As expected, we observe from Fig. 3 that the OWD of CoDel++ is 251 ms for slice 1, 658 ms for slice 2, and 700 ms for slice 3. It is also worth noting that thanks to the priority queuing, CoDel++ incurs less OWD for slice 1 than CoDel. However, since slice 2 and slice 3 have lower priority than slice 1, CoDel++ incurs higher OWD for slice 2 and slice 3 than for CoDel.

We then investigate the performance of the baseline scheme and the priority queuing scheme that exclude congestion control. Specifically, the baseline scheme performs worst among the studied schemes. As illustrated in Fig. 3, the OWD of the baseline scheme for each slice is varied from 624 ms to 2330 ms. This is expected as the baseline scheme does not consider the isolation between slices. Consequently, the results demonstrate the impact of network congestion on the latency for all slices. Whereas priority queuing is the second-worst scheme as it simply assigns the priority to each slice, i.e., slice 1 has the highest priority while slice 3 has the lowest priority. As expected, the results in Fig. 3 show that the OWD of the priority queuing scheme is 465 ms for slice 1, 495 ms for slice 2, and 1128 ms for slice 3. We also observe from Fig. 3, that the OWD for slice 3 tends to fluctuate due to its lowest priority highly.

In summary, CoDel++ can overcome congestion and provides strong isolation between slices. For the practical deployment, CoDel++ can assign high priority to slices for latency-sensitive use cases such as haptics.

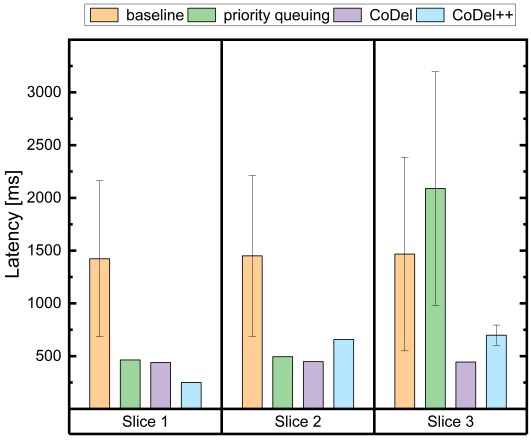


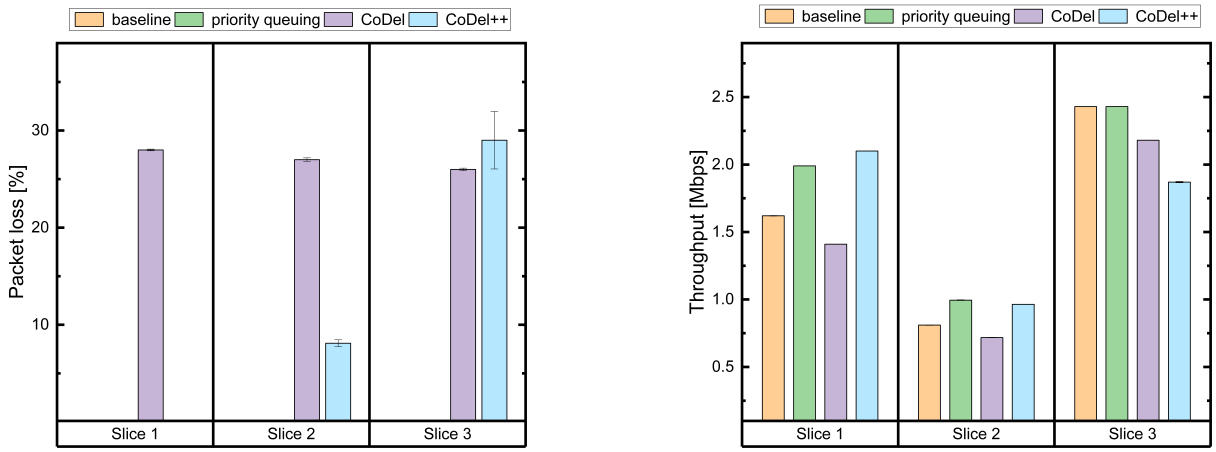
Fig. 3. Latency of each slice in all four scenarios

2) Packet loss: We examine the performance of the studied schemes for different slices in terms of packet loss. Fig. 4a plots the packet loss of the studied schemes for different slices. During the simulation, we observed that the total number of packets for all incoming data streams is the maximum queue depth for the baseline and the priority queuing scheme. As expected, Fig. 4a shows that the baseline scheme and the priority queuing scheme incur no packet loss for all slices. Meanwhile, CoDel and CoDel++ incur packet loss due to the use of the congestion control algorithm. Specifically, the packet loss of CoDel is 28% for slice 1, 27% for slice 2, and 26% for slice 3. The packet losses of CoDel++ for slice 1, slice 2, and slice 3 are in turns 0%, 8.1%, and 27%. As can be seen, CoDel++ outperforms CoDel for slice 1 and slice 2 in terms of packet loss. Notably, compared to CoDel, when combined with the priority queuing, CoDel++ incurs no packet loss for slice 1 and only 8.1% packet loss for slice 2. Since CoDel++ assigns the lowest priority to slice 3, we observe that CoDel++ incurs 29% packet loss for slice 3.

In summary, the baseline scheme and the priority queuing scheme outperform CoDel and CoDel++ in terms of packet loss. However, in case of buffer overflow, the baseline scheme and the priority queuing scheme have to drop packets due to the lack of a congestion control algorithm. Notably, CoDel++ tends to have more flexibility than CoDel. Specifically, CoDel++ can allocate slices with low priority to use cases that can tolerate packet loss, such as video streaming. Meanwhile, high-prior slices are suitable for haptic data streams, which are highly sensitive to delay and losses.

D. Throughput

We study the performance of the schemes for different slices in terms of throughput. Fig. 4b plots the throughput of the studied schemes measured in 10 seconds. It is worth noting



(a) (b)

Fig. 4. Packet loss (a) and Throughput (b) of each slice in all four scenarios.

that priority queuing shows a significant impact on throughput. Specifically, we observe from Fig. 4b that the priority queuing scheme achieves higher throughput for slice 1 and slice 2 than the baseline scheme. Since slice 3 has the lowest priority, the priority queuing scheme achieves similar throughput compared to the baseline scheme. Meanwhile, thanks to the priority queuing, CoDel++ performs better than CoDel for slice 1 and slice 2 in terms of throughput. For slice 3, which has the lowest priority, CoDel++ achieves lower throughput than CoDel due to the packet loss increase. We also remark on the impact of the congestion control algorithm on the throughput. The results in Fig. 4b show that CoDel performs worse than the baseline scheme and the queuing scheme for all slices in terms of throughput. For instance, the throughput of CoDel is 1.41 Mbps for slice 1, 0.72 Mbps for slice 2, and 2.18 Mbps for slice 3. Meanwhile, the throughput of the baseline scheme is 1.6 Mbps for slice 1, 0.81 Mbps for slice 2, and 2.43 Mbps for slice 3. This is because CoDel tends to drop packets to alleviate the congestion, consequently lowering the throughput.

In summary, we observe the impact of priority queuing and congestion control on the throughput of the studied schemes. Specifically, priority queuing could improve the throughput of the schemes that rely on the congestion control algorithm.

V. Conclusion & Future Work

Future communication networks need to support multiple modalities with different latency and packet loss requirements to enable perceived real-time interactions between humans and robots. However, state-of-the-art congestion control algorithms, CoDel, apply the same policy for all modalities. This work studies the impact of priority queueing in combination with congestion on multiple modalities. We leverage programmable data planes, which can be programmed with P4, allowing for implementing AQM and congestion control algorithms in practical settings. Evaluation results show that combining priority queuing and CoDel (or CoDel++) produces superior results for high-priority data streams. CoDel++ reduces latency by roughly 60% and packet loss by half compared to individual exploitation of AQM or CoDel individually.

This work suggests promising future studies, such as evaluating the performance of CoDel++ in practical testbeds exploiting Tofino switches. We also plan to assess the impacts of different priorities in finer granularities. Last but not least, we will investigate more comprehensive topologies resembling real-world networks and traffic conditions more closely.

Acknowledgment

This work is funded in part by the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden and the Federal Ministry of Education and Research of Germany in the programme of "Souverän. Digital. Vernetzt." - Joint project 6G-life - projectID: 16KISK001K.

The authors would like to thank anonymous reviewers for constructive and valuable feedback.

References

[1] E. Steinbach, S. Hirche, M. Ernst, F. Brandi, R. Chaudhari, J. Kammerl, and I. Vittorias, "Haptic communications," *Proceedings of the IEEE*, vol. 100, no. 4, pp. 937–956, 2012.

[2] F. H. Fitzek, S.-C. Li, S. Speidel, and T. Strufe, "Chapter 1 - tactile internet with human-in-the-loop: New frontiers of transdisciplinary research," in *Tactile Internet*, F. H. Fitzek, S.-C. Li, S. Speidel, T. Strufe, M. Simsek, and M. Reisslein, Eds. Academic Press, 2021, pp. 1–19. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128213438000101

[3] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[4] K. Nichols and V. Jacobson, "Controlling queue delay," *Commun. ACM*, vol. 55, no. 7, p. 42–50, jul 2012. [Online]. Available: https://doi.org/10.1145/2209249.2209264

[5] C. Ford, "Lstfcodel: Codel with lstf-style priority queuing," 2020.

[6] J. F. Kurose and K. W. Ross, *Computer networking: Complete package*, 3rd ed. Boston: Addison-Wesley, 2006.

[7] Y. Gong, D. Rossi, C. Testa, S. Valenti, and M. D. Täht, "Fighting the bufferbloat: On the coexistence of aqm and low priority congestion control," in *2013 Proceedings IEEE INFOCOM*, 2013, pp. 3291–3296.

[8] C. Brandauer, G. Iannaccone, C. Diot, T. Ziegler, S. Fdida, and M. May, "Comparison of tail drop and active queue management performance for bulk-data and web-like internet traffic," in *Proceedings. Sixth IEEE Symposium on Computers and Communications*, 2001, pp. 122–129.

[9] N. G. Singh and Y. Nagar, "Comparing different active queue management techniques," vol. 4, no. 5, 2015, pp. 50–56.

[10] T. Ott, T. Lakshman, and L. Wong, "Sred: stabilized red," in *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, vol. 3, 1999, pp. 1346–1355 vol.3.

[11] J. Aweya, M. Ouellette, and D. Montuno, "A control theoretic approach to active queue management," *Computer Networks*, vol. 36, pp. 203–235, 07 2001.

[12] S. Floyd, R. Gummadi, S. Shenker *et al.*, "Adaptive red: An algorithm for increasing the robustness of red's active queue management," 2001.

[13] D. M. Raghuvanshi, B. Annappa, and M. P. Tahiliani, "On the effectiveness of codel for active queue management," in *2013 Third International Conference on Advanced Computing and Communication Technologies (ACCT)*, 2013, pp. 107–114.

[14] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management." [Online]. Available: http://dx.doi.org/10.17487/RFC8289

[15] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[16] R. Kundel, A. Rizk, J. Blendin, B. Koldehofe, R. Hark, and R. Steinmetz, "P4-CoDel: Experiences on Programmable Data Plane Hardware."

[17] E. Hauser, M. Simon, H. Stubbe, S. Gallenmüller, and G. Carle, "Slicing networks with p4 hardware and software targets," in *Proceedings of the ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases*, ser. 5G-MeMU '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 36–42.

[18] Y.-W. Chen, L.-H. Yen, W.-C. Wang, C.-A. Chuang, Y.-S. Liu, and C.-C. Tseng, "P4-Enabled Bandwidth Management," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE, 2019, pp. 1–5. [Online]. Available: http://dx.doi.org/10.23919/APNOMS.2019.8892909

[19] H. Harkous, C. Papagianni, K. de Schepper, M. Jarschel, M. Dimolianis, and R. Pries, "Virtual Queues for P4: A Poor Man's Programmable Traffic Manager," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2860–2872, 2021.

[20] R. Kundel, "P4 Codel Implementation," accessed: 2022-11-30. [Online]. Available: https://github.com/ralfkundel/p4-codel

[21] "Software-Defined Networks: A System Approach: Chapter 4: Bare-Metal Switches," 2022, accessed: 2022-06-17. [Online]. Available: https://sdn.systemsapproach.org/switch.html

[22] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, and R. Steinmetz, "P4-CoDel: Active Queue Management in Programmable Data Planes," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2018, pp. 1–4.

[23] G. Gombos, M. Mouw, S. Laki, C. Papagianni, and K. De Schepper, "Active queue management on the tofino programmable switch: The (dual)pi2 case," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 1685–1691.

[24] J. Liu, J. Huang, Z. Li, Y. Li, J. Wang, and T. He, "Achieving per-flow fairness and high utilization with limited priority queues in data center," *IEEE/ACM Transactions on Networking*, vol. 30, no. 5, pp. 2374–2387, 2022.

[25] C. Pan, S. Zhang, C. Zhao, H. Shi, Z. Kong, and X. Cui, "A novel active queue management algorithm based on average queue length change rate," *IEEE Access*, vol. 10, pp. 75 558–75 570, 2022.

[26] J. Ktz, "Measuring Network Throughput using NetPerf & iPerf," accessed: 2022-06-18. [Online]. Available: MeasuringNetworkThroughputusingNetPerf&iPerf

[27] "Mininet," accessed: 2022-11-30. [Online]. Available: http://mininet.org/

[28] "P4-Utils," accessed: 2022-11-30. [Online]. Available: https://nsg-ethz.github.io/p4-utils/usage.html

[29] "P4 behavioral model: The reference P4 software switch," accessed: 2022-11-24. [Online]. Available: https://github.com/p4lang/behavioral-model

[30] "p4c: The reference P4 compiler," accessed: 2022-11-30. [Online]. Available: https://github.com/p4lang/p4c