Fast network congestion detection and avoidance using P4

Turkovic, Belma; Kuipers, Fernando; van Adrichem, Niels; Langendoen, Koen

DOI
10.1145/3229574.3229581

Publication date
2018

Document Version
Accepted author manuscript

Published in
NEAT'18

Fast network congestion detection and avoidance using P4

Belma Turkovic
Delft University of Technology
B.Turkovic-2@tudelft.nl

Fernando Kuipers
Delft University of Technology
F.A.Kuipers@tudelft.nl

Niels van Adrichem
TNO
niels.vanadrichem@tno.nl

Koen Langendoen
Delft University of Technology
K.G.Langendoen@tudelft.nl

ABSTRACT

Along with exciting visions for 5G communications and the Tactile Internet, the networking requirement of attaining extremely low end-to-end latency has appeared. While network devices are typically equipped with buffers to counteract packet loss caused by short-lived traffic bursts, the more those buffers get filled, the more delay is added to every packet passing through.

In this paper, we develop congestion avoidance methods that harness the power of fully programmable data-planes. The corresponding programmable switches, through languages such as P4, can be programmed to gather and react to important packet meta-data, such as queue load, while the data packets are being processed. In particular, we enable P4 switches to (1) track processing and queuing delays of latency-critical flows and (2) react immediately in the data-plane to congestion by rerouting the affected flows. Through a proof-of-concept implementation in emulation and on real hardware, we demonstrate that a data-plane approach reduces average and maximum delay, as well as jitter, when compared to non-programmable approaches.

CCS CONCEPTS

• **Networks → Data path algorithms**; *Programmable networks*;

KEYWORDS

Low latency, Programmable data-planes, Tactile Internet, 5G.

1 INTRODUCTION

For long, available network capacity has been the most important Quality-of-Service (QoS) parameter to optimize for. Recently, with the emergence of novel application domains such as the Tactile Internet – where the objective is to transport a sense of touch over the Internet – and supporting communications technologies such as 5G, low latency has also become a crucial QoS parameter. Tactile Internet applications need very low latency ($\approx 1ms$), low jitter, high bandwidth (in the order of Gbps), and high reliability [4, 6, 9].

Tactile Internet traffic could be very bursty, depending on the required modalities (audio, video, and/or haptic). While prediction algorithms might relax the latency requirement, consistent feedback and a maximum delay bound are necessary for a haptic system to be stable. Consequently, to minimize the end-to-end latency, packets of Tactile Internet flows should not be delayed on any node on the path nor be dropped by the network. This requires network nodes to be able to quickly detect and react to any changes in the network state, such as buffers filling up.

A packet typically encounters four types of network delay:

- Propagation delay: a function of the physical distance and propagation speed of a link.
- Transmission delay: a function of the size of a packet and the data rate of the link.
- Processing delay: the time required to inspect a packet header and determine its destination.
- Queuing delay: the amount of time a packet is waiting in a queue until it can be transmitted.

In contrast to the propagation and transmission delays, the processing and queuing delays depend on the amount of traffic and how it is handled in the network. As such, they may vary significantly and controlling and limiting them is of importance and therefore the main topic of this paper.

1.1 Problem definition

One of the most important factors that contributes to queuing delay is congestion, which occurs when a network node is trying to forward more data than the outgoing link can process.

Congestion control mechanisms of traditional transport protocols such as TCP detect congestion at the sender node and modify the sending rate accordingly. In the case of tactile traffic, such an approach is not feasible as it is not allowed to buffer or increase/decrease the rate at the tactile source. Furthermore, many congestion control algorithms only kick in after congestion has occurred and need at least one round-trip time (RTT) to react to the perceived congestion. Software-defined networking (SDN [8, 10]), as a new paradigm in networking, offers an alternative. Because every node in the network is controlled from software-based controllers, these controllers have a centralized view of the network and are able to react and adapt to changing network conditions faster. A common method to provide QoS in SDN is to implement virtual slicing of the available bandwidth on all the nodes on the path, reserving parts of it for different services, or to use priority queuing. But, as the required bandwidth can be in the order of a few Gbps [9], reserving the maximum required bandwidth for every flow is not scalable. Priority queuing, while minimizing queuing delay for the higher prioritized flows, can lead to starvation of flows and does not prevent congestion. In fact, high-priority flows will starve when congestion forces low-priority flows to occupy all available queue space. Alternatively, IEEE 802.1TSN works on standardizing specialized schedulers for Time-Sensitive Networks (TSN), such as time-aware traffic shapers [7], though those solutions require a closed-circuit network to operate.

There are many frameworks that use some form of QoS routing to find the path that satisfies different QoS requirements. However,

SDN frameworks from this group depend on some form of monitoring ([11, 12]). Incorrectly set monitoring intervals have direct influence on the usefulness of the gathered data as well as the number of probe packets sent. Additionally, after congestion is detected, a certain time is needed for the controller to recompute the path and reconfigure table entries before switching the flow to a better path. To avoid the aforementioned artifacts, the main problem to be solved is: *How to enable congestion control and avoidance in the forwarding nodes, instead of at the source or via a controller?*

1.2 Main contributions

In Section 2, we propose a hierarchical control model for latency-critical flows. Our solution contains a small program running directly on the switches, which has real-time access to latency monitoring data to quickly reroute traffic when degradation is detected.

In Section 3, we evaluate our solution both by emulation through software switches as well as with P4 hardware. We compare our solution against a congestion-agnostic approach as well as to congestion-avoidance approaches that make use of probing.

2 CONGESTION DETECTION AND AVOIDANCE IN THE DATAPLANE

If end-to-end delay, as well as jitter, needs to be kept under a certain threshold, the main challenge is to detect and react to any increase in delay when data is being processed at the switches and not (only) at the source. If every switch minimizes the total delay per node (for certain traffic flows) to a configurable value, bounding maximum end-to-end delay becomes feasible.

Recently, in the wake of SDN, programmable switches have appeared along with domain-specific programming languages such as P4 to program them [5]. P4 offers the possibility to gather and export important packet meta-data (timestamps from different stages of processing, queue depths, etc.) directly from the data-plane while the data-packets are being processed. To leverage this unique possibility of collecting packet meta-data, we propose a hierarchal architecture, as shown in Fig. 1, to detect and avoid congestion:

A local congestion control module at the P4 switches, as elaborated on in Sec. 2.1, monitors the state of all the low-latency flows, while a central controller configures the latency thresholds and other parameters.

2.1 Local control

A local Congestion Detection and Avoidance module, see Fig. 2, is developed to monitor the processing and queuing delays.

If the module determines that one of these delay components is increasing for a latency-critical flow, and congestion is likely to occur, it preemptively switches the traffic to a better backup path if it exists or signals to the previous node in the path that it is congested and that it should not forward any more packets belonging to that flow.

According to the P4 language specifications [1, 2], table entries at a switch cannot be modified without the intervention of the control-plane (controller). Thus, in order to achieve rerouting in the data-plane we are left with two choices: (1) add both entries (primary and backup) to a table and decide which rule to apply based on some meta-data stored in the registers, or (2) send packets

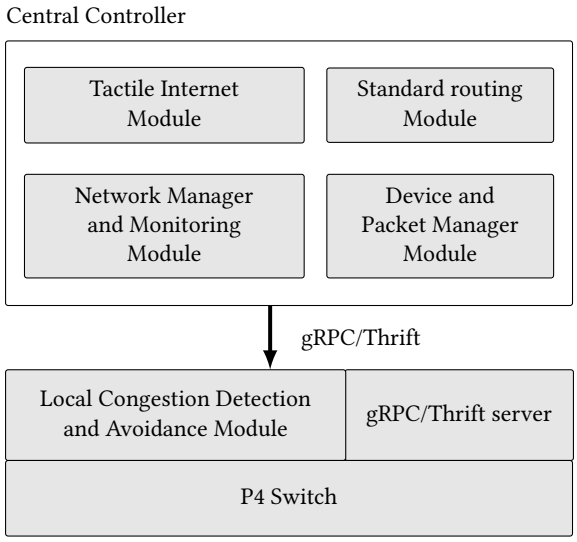


Figure 1: Hierarchical design of the control plane.

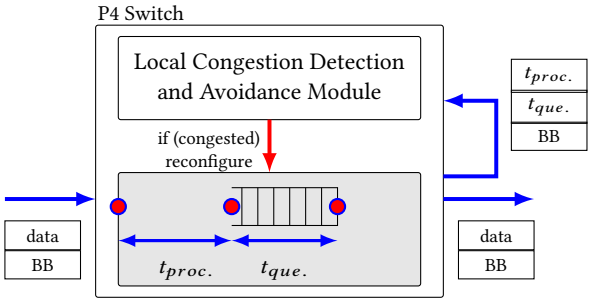


Figure 2: Detection of congestion in the data-plane. Every switch has a small congestion avoidance module, gathering statistics (processing and queuing delay).

or packet digest notifications to a local listener that tracks the flows and acts as a small local control-plane.

If we use meta-data and registers, the solution is applicable to all P4-capable hardware and all processing is done entirely in the data-plane. As a consequence processing delay per packet is increased, as more table entries and registers are needed to maintain accurate flow state in the data-plane. Additionally, updating registers per packet can lead to race conditions when packets from the same flow are processed in parallel. Since meta-data and register values affect table entries dynamically, table lookup caching, must be disabled.

Alternatively, if we use packet copies or packet digests, a local listener module (that is running on the same machine as the switch itself) can make local routing decisions based on the received data. The disadvantage is a slightly higher detection delay, since transmitting the digest packet to and processing it at the local control module takes additional time. Additionally, while digest notifications are very small (containing only relevant packet data and meta-data), the rate at which they are generated can be very high if we want to obtain delay information about every packet on the path. To avoid overloading the local control application module,

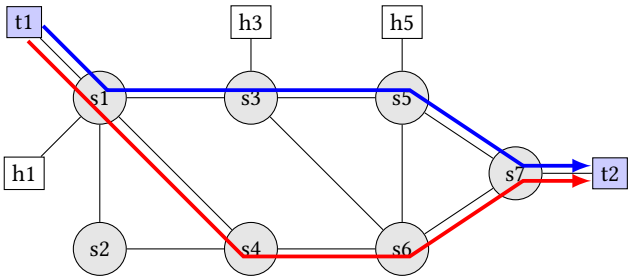


Figure 3: The blue line is the primary path and the red line is the backup path that traffic will take when any of the routers on the primary path detects congestion. To be effective, both paths have the same weight and are assumed equally stable.

we will shift the detection of the congestion to a P4 program and use digests to notify the local module about congestion only when the delay increases above a certain threshold.

The number of consecutive packets m with increased delay needed to signal congestion, as well as the threshold values for queuing t_q and processing t_p delays are configurable and depend on the type of hardware used as well as the sensitivity needed. On the one hand, if the thresholds are too small, the local control module might reroute traffic unnecessarily, potentially increasing the jitter as well as creating additional load on the central controller that needs to recalculate a new backup route, delete rules from the old primary path and install new backup rules. On the other hand, if the thresholds are too large, the control module might react too late, thus providing little increase in performance when compared to legacy solutions.

2.2 Rerouting example

In order to have adequate backup paths available, for every new tactile flow two paths that satisfy the latency requirement of that flow are calculated, as shown in Fig. 3. The blue path (s1-s3-s5-s7) is used as the primary path. The red path (s1-s4-s6-s7) is used as backup. Multiple primary and/or backup paths could also have been used, but we have opted for single paths, since it requires less processing (in terms of packet re-ordering) at the end-node.

In case an increase in queuing delay is detected on switch s1, the local congestion control program will change the output port for this flow to s4 (as shown in Fig. 4). Switch s1 is still used, as output ports and consequently output queues are different and not affected by the detected queue build-up. The tactile route is already configured on switches s4 and s6 and thus rerouting is achieved instantly. The local control programs at the switches determine the output ports based on input they receive from a central controller that tracks the link utilizations and delays on all the nodes in the network by sampling the network constantly.

In case switch s5 detects congestion (increased queuing delay on the tactile queue to s7 or increased processing delay) it has no better route configured and can thus only signal to its predecessor that the output link is congested. It thus sends a control message (congestion notification) to s3, who forwards it to s1. When s1 receives this message it will switch the affected tactile flow to s4 (as shown in Fig. 5).

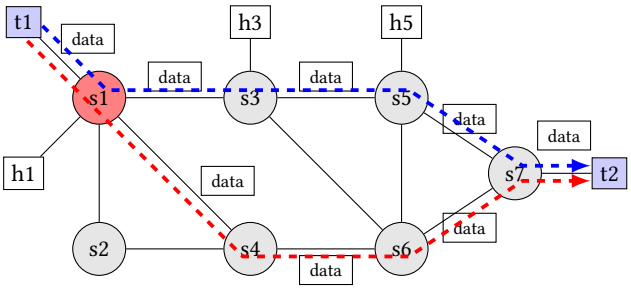


Figure 4: If switch s1 is congested, it already knows a better backup route. S1 will consequently reroute the traffic to s4.

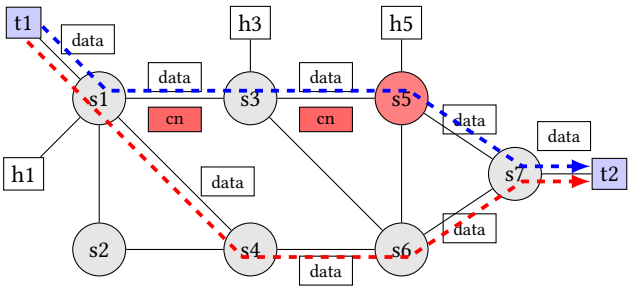


Figure 5: If switch s5 is congested, no better route can be found to t2. In this case, s5 will send a packet to s3 informing s3 not to send packets through s5. S3 forwards this control packet to s1, who will consequently reroute the traffic to s4.

It is important to notice that links of the backup path are disjoint from the links of the primary path after s1 (the switch that can actually perform the fail-over). The paths are calculated this way to prevent the backup path from forwarding the traffic to the same congested link as the primary path. The central controller, which has knowledge about the whole network, computes these paths periodically, based on the current network state. When a flow is switched to a backup route, that route becomes the new primary path and a new backup must be computed and installed.

3 EVALUATION USING EMULATION

We have evaluated our solution, via the set-up described in Sec. 3.1, using the Mininet emulation environment with the P4 software switch (behavioral model, nicknamed *bmv2* [3]).

3.1 Experiment setup

Multiple flows were generated and RTT, maximum RTT, packet loss, as well as ingress processing, queuing, jitter, detection and reaction delays were measured. Per tactile flow, one primary and one backup route were configured. Additional traffic was generated to create congestion on different intermediate nodes on the primary route. Each tactile traffic trace was 15 seconds long, and these scenarios were repeated 40 times.

We varied the detection threshold for processing and queuing delays, as well as the number of consecutive packets m that need to have an increased delay in order to detect congestion. Scenario

DataplaneX_m represents a scenario where the thresholds for processing and queuing delays were X times the processing and queuing delays on the switch if no additional load was generated and m is as defined before.

Our approach was compared to (1) an approach that uses no congestion detection and never recomputes paths (scenario No CC), which is mimicking traditional routing protocols such as OSPF and (2) an SDN-like approach that uses a centralized controller and periodically sends probe packets (scenarios Probing$X$sec), to determine the current network state and detect congestion. We used different monitoring intervals, namely 1, 2, and 5 sec.

3.2 Mininet results

We have used the network topology displayed in Fig. 3. The rate of all the bmv2 output queues was limited to 170.000 pkts/s ($\approx 200Mbps$) in order to make sure that there would be a queue build-up. With this configuration, as the packet arrival rate is smaller than what the bmv2 ingress pipeline can process ($\approx 1Gbps$ on a server in our testbed), the bandwidth, and not the processing is the bottleneck. If the rate of the output queues is not limited, when the maximum throughput is reached, packets are dropped before the ingress pipeline, and there is no queue build-up, since the egress pipeline is usually faster than that of the ingress in bmv2.

In our scenario, 8.000 packets per second ($\approx 4Mbps$ with a packet size of 64B) were injected by the tactile flow that we were interested in. If the amount of additional traffic was below the configured bottleneck bandwidth of $\approx 200Mbps$, the switches could process the low-latency data at line rate (Fig. 6). When the volume of additional traffic approached 200 Mbps, the delay on node s3 increased, as the total amount of traffic exceeded the configured rate of the output queue. This was also the point were all the evaluated approaches correctly detected congestion and reconfigured the path for the tactile data.

Detection time: In the probing scenarios, as the controller uses increased delay of probe packets as an indication of congestion, the smaller the probing interval, the faster the controller was able to detect congestion, as shown in Fig. 6c. As the volume of additional traffic increased, the number of dropped probe packets, as well as the maximum delay, increased as well. In these scenarios, when no probe was returned within the probing interval, the controller assumed that the packet was lost and the link congested. This is why the detection delay in Fig. 6c is higher than expected (half the monitoring interval). By comparing the values for the maximum detection delay, we observed that in the worst case it is approximately two times the value of the probing interval, which corresponds to one probe packet being sent immediately after congestion (in the queue build-up phase) and the subsequent packet being lost. Thus, the controller needed to wait for the timeout value (one monitoring interval) to expire.

In case the detection was done using the measurements in the data-plane, the controller was always able to detect the changes very fast, by observing the data itself independently of the probe packets that were sent. The advantage of this approach is especially noticeable when detection time is compared to other approaches, as shown in Fig. 6c. An increased number of subsequent packets m (Dataplane12_20) increases detection delay. However, this increase is very small when compared to scenarios Probing$X$sec.

Reaction time: After detecting congestion, in case of the probing scenarios, the controller needed to find a new route and install new table entries starting from the end of the path in order to minimize the number of dropped packets. After traffic was switched, some packets were still present in the queues of the congested node. Consequently, packets arrived in the wrong order at the endpoints.

All data-plane schemes only needed to update one table entry. The switches could immediately forward traffic on the new path and thus the total time needed to switch the traffic was minimized.

Delay and jitter measurements: The data-plane schemes, as a consequence of fast detection, had the lowest average and maximum delay, as can be seen in Fig. 6e. Increasing the number of subsequent packets m has a negative influence on the maximum delay, as well as maximum jitter, especially in case of very high additional traffic.

Average loss: In the case of no congestion control (scenario No CC), packets were queued until the buffer limit on s3 was reached, causing an increased number of dropped packets as can be seen in Fig. 6b. All probing scenarios were able to detect and reduce the number of dropped packets. By comparing the loss values, we can see that data-plane approach was the only one that could keep the loss value at 0%. For the probing solutions the loss increased with the amount of additional traffic, due to faster overruns of buffers.

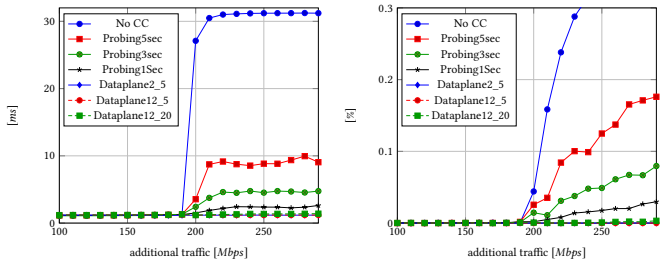
Artifacts caused by the environment: One of the identified problems was that, depending on the configured threshold for the detection in the data-plane, the probability of false negatives was significant (scenario Dataplane2_5). In these scenarios, although the threshold was set to twice the value of the queuing delay when no additional traffic was generated, switches detected congestion every time. By increasing the value of the threshold, or the number of subsequent packets needed, the value of false positives can be reduced, as shown in Fig. 6d, while maintaining the QoS parameters at almost the same level.

4 PROOF OF CONCEPT USING P4 HARDWARE

We have built a proof of concept using our P4 hardware testbed that consists of physical general-purpose servers enhanced with smart network interface cards (Netronome Agilio CX 2x10GbE), which were connected as shown in Fig. 7. All the servers used Thrift RPC as the control interface and ran Ubuntu with kernel version 4.10.

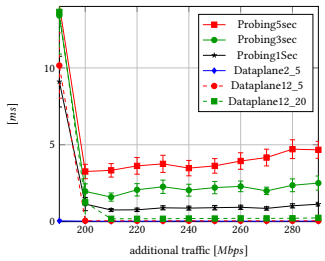
Two different data-plane approaches were evaluated. The first one (DP_direct) did all the processing in the data-plane, while the second one (DP_listener) implemented the detection of delay increase in the data-plane and did all the subsequent processing of the notifications in the local digest listener module. Our approach was compared to an approach that does no congestion control (scenario No CC, as in Sec. 3) as well as to an approach that uses periodic sampling of the current state of the network stored in switch registers (Probing1sec-5sec). All scenarios were repeated 50 times.

A tactile flow was generated between switches s1 and s4, while additional traffic was passing between hosts h1-h4 and hr, generating congestion on the output port of switch s2. The tactile flow had a throughput of 20 kpps ($\approx 240Mbps$ with packet size of

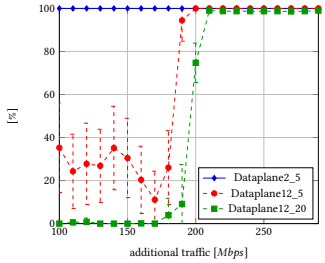


(a) Average RTT for tactile data.

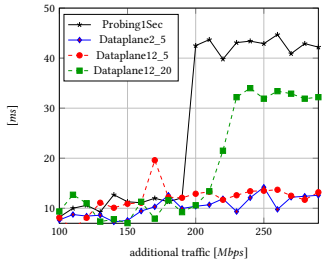
(b) Average loss for tactile data.



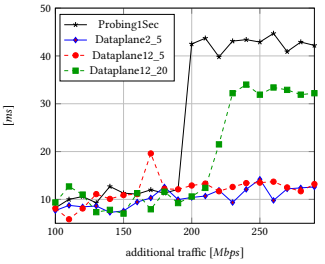
(c) Congestion detection + reaction delay.



(d) Percentage of tactile data that were rerouted to the backup path.



(e) Maximum RTT for tactile data.



(f) Comparison of maximum variation in RTT.

Figure 6: Mininet scenario (Confidence intervals 90%). Comparison of different QoS parameters for different schemes when congestion is present.

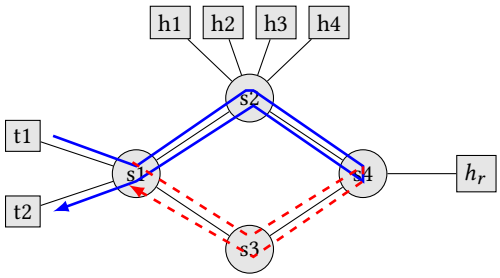


Figure 7: Hardware topology.

1500 B), while the additional traffic had a throughput of 1.5 Mpps (where the packet size varied between 64 B and 1100 B), creating load in the range of ≈ 750 Mbps to ≈ 13 Gbps. The first and last second of the trace were not taken into account for latency and jitter measurements and additional traffic started 2s after the tactile traffic in order to observe queue build-up. To achieve high accuracy (nanosecond range), as well as to limit the influence of external factors (e.g., processing in the driver, kernel, etc.), latency was measured in the data-plane at switch s1. Every tactile packet that was processed was equipped with an additional header field storing a 64-bit ingress time-stamp (when the packet was received from t1) or an egress time-stamp (when the packet was forwarded to t2). Since there is no external syncing between the switches, tactile traffic was routed back from s4 to switch s1, which inserted both timestamps, as shown in Fig. 7.

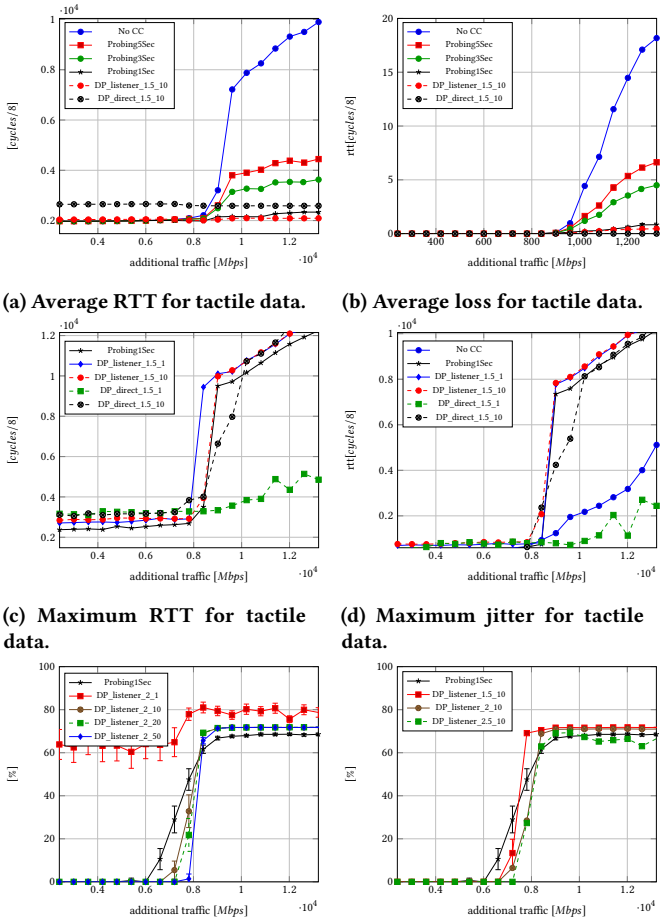
4.1 Netronome Agilio CX SmartNIC results

We encountered several limitations when we evaluated our scheme using the above-mentioned testbed. In an initial experiment, while measuring the ingress and egress processing delays, the delay between these two stages (which should represent queuing delay) was constant, even when the switch was congested and the total end-to-end delay increased. Because we were unable to obtain queuing delay information directly from the P4 program, we measured the total delay on the switch (from the ingress MAC component to the egress MAC component). An ingress time-stamp (the time in nanoseconds when the ingress MAC component receives the packet) was attached to the packet data structure while the packet was being processed at the card and could be inserted by the P4 program itself. In order to get the egress time-stamp, we added a special 32-bit header to the start of the packet. When the egress MAC component of the SmartNIC receives this special header it attaches the egress counter-value "time-stamp" and forwards the packet to the next switch. Since no external syncing is implemented, counter values can only be used for latency measurements inside one card. A subsequent switch in the path keeps track of the difference between these values in a register and based on that value decides whether the previous switch is congested or not. If it determines that the previous switch is congested it will send a congestion notification back, that, when received by s1, will trigger the rerouting to the backup path (s1-s3-s4). Thus, the detection of the congestion was shifted by one node, increasing the reaction time when compared to the emulated environment.

Measurements shown in Fig. 8 demonstrate a functional proof of concept. All the evaluated data-plane approaches, DP_listener and DP_direct, outperformed the other analyzed approaches by keeping all the analyzed QoS parameters on par with scenarios where no congestion was present. We have plotted only DP_direct_1.5_10 and DP_listener_1.5_10 in Figures 8a and 8b, but all other analyzed data-plane scenarios had similar performance.

Average and maximum delay: When the switches were not congested, the data-plane approaches, as a consequence of additional processing, had higher average and maximum delay than the other evaluated approaches (from ≈ 1, 900 in scenario No CC to ≈ 2, 000 for Dataplane_listener and ≈ 2, 500[$cycles/8$] in case of Dataplane_direct). The significant increase in average as well as maximum delay for the Dataplane_direct scenario is a consequence of a more complex data-plane implementation, since multiple tables and registers are needed to keep the per-flow state.

For higher volumes of additional traffic, only direct data-plane approaches were able to keep the maximum delay at the same level



(a) Average RTT for tactile data.

(b) Average loss for tactile data.

(c) Maximum RTT for tactile data.

(d) Maximum jitter for tactile data.

(e) The influence of the number of packets m used to detect congestion on the average percentage of packets that were processed on the backup path.

(f) The influence of detection threshold ($t_p + t_q$) on the average percentage of packets that were processed on the backup path.

Figure 8: Netronome SmartNIC scenario (Confidence intervals 90%). Comparison of different QoS parameters for different schemes when additional traffic is generated to create congestion at node s2.

as before, as shown in Fig. 8a. Switch s1 was the one that rerouted the traffic, causing delay between detection and reaction. Even the data-plane approaches were unable to keep the maximum latency value below a certain threshold, especially for higher m.

Maximum jitter was lowest for No CC approach and DP_direct (Fig. 8d). The relatively high jitter for the other approaches is a consequence of switching paths. The first packet that is processed on the backup path has a very low RTT compared to the ones that are still processed by the congested nodes.

Congestion detection and sensitivity Increasing the detection threshold, shown in Fig. 8f, has a negative influence as we miss the start queue buildup phase and, consequently, more packets are affected by congestion. By decreasing the threshold, even when the value of additional traffic was not high enough to cause congestion, all analyzed approaches (including the Probing scenarios) detected it. In cases when both primary and backup paths have high link utilizations, this behavior may lead to too many recalculations and path switching, which would degrade the overall performance. This can be resolved by increasing the number of packets used to detect congestion, as shown in Fig. 8e.

5 CONCLUSION

To quickly detect and avoid congestion within a network, we have proposed a P4-based technique that enables measuring delays and rerouting in the data-plane. Our approach offers two main advantages. First, the detection time is reduced and congestion is detected per flow. Thus, only the affected flows are rerouted and QoS degradation of other flows is avoided. Second, after detection, the reaction time is minimized as a local controller, based on input from a central controller, intervenes by configuring a better route. Therefore, no new flow rules need to be installed and the load on the central controller is reduced.

We encountered some limitations with the evaluation of our solution using Netronome P4 SmartNICs, such as a limit on range matching, a performance penalty due to disabled caching, as well as lack of information about the queuing delay of the current switch. Nonetheless, we were able to show the feasibility of our solution in both emulated and physical networks.

While the presented approach requires specialized hardware (P4-capable switches), in a hybrid network where only some nodes can be programmed, detection time as well as reaction time to congestion might still be reduced using this scheme, when the nodes are placed at crucial points in the network such as the network edge. Additionally, our solution can easily be extended to a solution that uses bandwidth reservation and/or priority queuing.

REFERENCES

[1] P4 14 language specification. https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf. Accessed: 19-03-2018.

[2] P4 16 language specification. https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html. Accessed: 19-03-2018.

[3] P4 behavioral model. https://github.com/p4lang/behavioral-model. Accessed: 19-03-2018.

[4] Berg, D. V. D., Glans, R., Koning, D. D., Kuipers, F. A., Lugtenburg, J., Polachan, K., Venkata, P. T., Singh, C., Turkovic, B., and Wijk, B. V. Challenges in haptic communications over the tactile internet. *IEEE Access 5* (2017), 23502–23518.

[5] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 87–95.

[6] Fettweis, G. The tactile internet: Applications & challenges. *IEEE Vehic. Tech. Mag. 9*, 1 (March 2014), 64 – 70.

[7] Maxim, D., and Song, Y.-Q. Delay analysis of avb traffic in time-sensitive networks (tsn). In *Proceedings of the 25th International Conference on Real-Time Networks and Systems* (2017), ACM, pp. 18–27.

[8] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev. 38*, 2 (Mar. 2008), 69–74.

[9] Seam, A., Poll, A., Wright, R., Mueller, J., and Hoodbhoy, F. Enabling mobile augmented and virtual reality with 5g networks, January 2017.

[10] Sezer, S., Scott-Hayward, S., Chouhan, P., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., and Rao, N. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine 51*, 7 (2013), 36–43.

[11] Shu, Z., Wan, J., Lin, J., Wang, S., Li, D., Rho, S., and Yang, C. Traffic engineering in software-defined networking: Measurement and management. *IEEE Access 4* (2016), 3246–3256.

[12] van Adrichem, N. L. M., Doerr, C., and Kuipers, F. A. Opennetmon: Network monitoring in openflow software-defined networks. In *2014 IEEE Network*

Operations and Management Symposium (NOMS) (May 2014), pp. 1–8.