Per Priority Data Rate Measurement in Data Plane

Habib Mostafaei
Eindhoven University of Technology

Georgios Smaragdakis
Delft University of Technology

ABSTRACT

Many applications, such as video streaming, congestion control, and server selection, can benefit when the data rate of different priority groups between two endpoints is accurately estimated over the end-to-end path. With the introduction of programmable networks, e.g., P4, it is now possible to offload the measurements to the data plane of intermediate devices. Recently, tools have been developed to react to changes in available bandwidth, but a tool to accurately estimate end-to-end per-priority data rates needs to be added. This motivates us to design and implement a new end-to-end and per-priority data rate estimation tool, PrioMeter. PrioMeter can accurately report the data rate per priority group of flows in programmable networks using high-precision timestamps for arbitrary traffic scales. PrioMeter leverages two primitives: quantization and truncation to achieve its goals. We implement PrioMeter in P4 and test it on BMv2 switches, and our preliminary results using NS3 simulations show that it can accurately estimate the data rate of different priority flows with minimal overhead.

CCS CONCEPTS

• **Networks** → **Programmable networks**; **Network monitoring**; **Network measurement**; **Network management**.

KEYWORDS

Programmable networks, network measurement.

ACM Reference Format:
Habib Mostafaei and Georgios Smaragdakis. 2023. Per Priority Data Rate Measurement in Data Plane. In *Proceedings of the 6th European P4 Workshop (EuroP4 '23), December 8, 2023, Paris, France.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3630047.3630199

1 INTRODUCTION

With the rapid growth of modern communication networks and services running on them, e.g., video streaming, network operators need to measure the traffic of different services to meet strict Quality of Service (QoS) requirements or fairly share the bandwidth among users [14]. The available bandwidth measurement is a common technique to estimate the residual capacity of the network to offer new services. For instance, content providers leverage the end-to-end available bandwidth estimation information to better balance the network load [20]. Moreover, knowledge about the data rate of individual flows, or per set of flows that belong to the same *priority groups (PG)* can enhance the performance of a network by classifying the packets and by prioritizing user traffic to achieve good user experience and engagement and increase application revenues [11]. Performance of applications running in datacenters can also be enhanced when accurate end-to-end information about the data rate per priority flow group is accurately estimated [15].

Many techniques and tools exist to measure the end-to-end available bandwidth such as pathLoad [11], pathChirp [18], IGI [9], iPerf [10], to name a few. The common approach among these techniques and tools is to leverage either probe packets (packet trains) or bulk transfer statistics to measure the available end-to-end bandwidth. Unfortunately, none of them can be used to accurately estimate the end-to-end data rate per group of flows.

In this paper, we argue that with the advancements in programmable networks and the use of P4 [8], it is now possible to have access to information about the status of intermediate network devices by utilizing widely available P4-enabled telemetry tools in commodity P4 switches. We design, develop, and evaluate a new per priority data rate estimation tool PrioMeter in programmable networks. PrioMeter uses high-resolution timestamps provided by the P4 programmable switches to measure the amount of transferred data in the data plane per priority groups of flows. PrioMeter supports finer-grained measurements, which enables network operators to allocate resources more efficiently.

Although readily available congestion control tools, e.g., HPCC [13] or PINT [6], provide information about the queue size and utilization of individual links, they do not report on the end-to-end data rate per group of flows with the same priority. With PrioMeter, we provide the first end-to-end data rate estimation per priority group of flows. We believe that PrioMeter can unleash the full potential of programmable networks and satisfy hard application requirements that run on top of programmable networks.

PrioMeter adds a fixed header field to the packet's header fields to carry the flow priority group rate on each path. However, carrying the data rate of different priorities over the network, especially in the datacenter networks, adds overhead to the network and negatively affects the flow completion time (FCT). To minimize the overhead, PrioMeter leverages two primitives: *quantization* and *truncation*. The first primitive compacts the measured data rate per priority on the data plane of programmable devices. The network operators can control the granularity of the measurements by deciding on the number of bins (quantization) to keep the data rate. The second primitive truncates the quantized values to a representative data rate per priority. With these two primitives, our evaluation on a large-scale datacenter network shows PrioMeter has comparable FCT slowdown with the state-of-the-art tools such as HPCC [13]. Our contributions can be summarized as follows:

- We design, implement, and evaluate PrioMeter, a system that utilizes high-resolution timestamps of P4-enabled switches to accurately estimate data rate of per flow priority group.
- We report that PrioMeter's overhead is comparable with other data plane-based tools with additional feature of reporting per priority data rates.

EuroP4 '23, December 8, 2023, Paris, France

ACM ISBN 979-8-4007-0446-8/23/12.
https://doi.org/10.1145/3630047.3630199

- We make the implementation of PrioMeter publicly available in [3].

The rest of this paper is organized as follows. Section 2 surveys the existing works. We present the design of PrioMeter in Section 3. Section 4 reports on the performance of our empirical evaluation under different workloads and we conclude in Section 5.

2 RELATED WORK

This section presents the state-of-the-art available bandwidth estimation techniques and tools.

2.1 Bulk Transferred-based Measurements

Some of the existing available bandwidth estimating tools, e.g., iPerf [10], TTCP [19], and NetPerf [16] measure the end-to-end available bandwidth on a path by transferring large packets. Notice that the use of the above tools adds significant traffic overhead on the network [17] and they fail in measuring the data rate of each PG of flows.

2.2 Probe Packet-based Measurements

Most of the available bandwidth estimating tools, e.g., Pathload [11], PathChirp [18], IGI/PTR [9], and CAPSET [12] send per path and periodically a small number of probes to reduce the measurement load. All the above tools do not have access to the state of individual network components and have no support to measure per PG data rate.

2.3 Data Plane-based Solutions

HPCC [13], PINT [6], and Bolt [5] utilize in-network telemetry to get the link utilization information to control congestion without being able to meter the data rate of each PG. However, HPCC, PINT, and Bolt are not designed to report information about the end-to-end data rate per PG on a network path.

3 THE PrioMeter SYSTEM

This section presents the architecture of our system, PrioMeter, describes how it leverages high-precision timestamps to measure the data rate of PGs, and defines per-priority data rate aggregation. First, we present the aggregation technique used in our approach to collect the data rate for each PG. Then, we sketch how PrioMeter measures the data rate of different PGs.

3.1 PrioMeter Aggregation

Per-priority aggregation summarizes the data rate value of each PG among links from the source to the destination in a path. We formally define it as follows.

Consider a network with a set of devices e.g., switches or routers, and links. Also, consider a path p between a *sender* and a *receiver* in this network. We assume $H = \{h_1, h_2, \ldots, h_n\}$ is a set of hops for path p that does not change during the measurement time window j. Each hop h_1 has a set of flows with different PGs, $\rho = \{pg_1, pg_2, \ldots, pg_k\}$.

For each hop like h_1, we denote the set $\{r_{h_1}^{pg_1}, r_{h_1}^{pg2}, \ldots, r_{h_1}^{pg_k}\}$ as the current data rates for each *PG* in the measurement time window j. For a specific pg, e.g., pg_j, on path p, the data rate is given by:

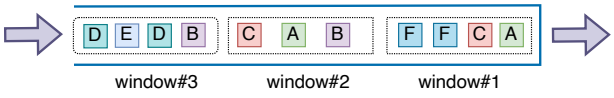


Figure 1: Snapshots of three sliding windows of PrioMeter.

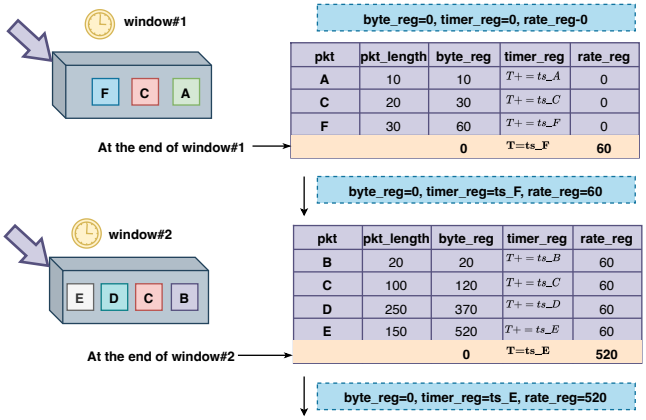


Figure 2: An example of data rate computation of PrioMeter for two different time window. PrioMeter receives three packets in each time window and updates the values of the registers accordingly.

$$R_{\text{pg}_j} = \max_{h_i \in H} r_{h_i}^{\text{pg}_j} \tag{1}$$

where n is the total number of hops, and k is the total number of PGs. Herein, the number of hops for each path can vary depending on the network topology.

3.2 Priority-based Aggregation

PrioMeter performs measurements in time windows. We define a threshold value for the length of the window to compute the amount of transferred data on each port. Figure 1 shows an example of measurement with three windows with a different number of incoming packets in each, but with the same PG to simplify the illustration. Each colored box in Figure 1 indicates a packet, and the switch can receive multiple packets of the same type in a measurement window. We now detail the measurement mechanism.

We introduce two different registers to measure the transmitted traffic over a port, namely, `bytes_reg` and `rate_reg`. We also define `timer_reg` register to measure the time window length. We set a fixed value as the threshold to reset the values of the three registers. We use `bytes_reg` register to count the amount of transmitted bytes per PG per port. `rate_reg` register maintains the latest data rate from each port per PG. P4 registers are stateful memories that can maintain user-defined data on the P4 switches.

When a packet arrives at a device, it checks the size of the packet and updates the value of the `bytes_reg` register. PrioMeter does this measurement for all the packets crossing each egress port. This measurement implies that the number of cells in the `bytes_reg` register is equal to the number of the egress ports of the switch times

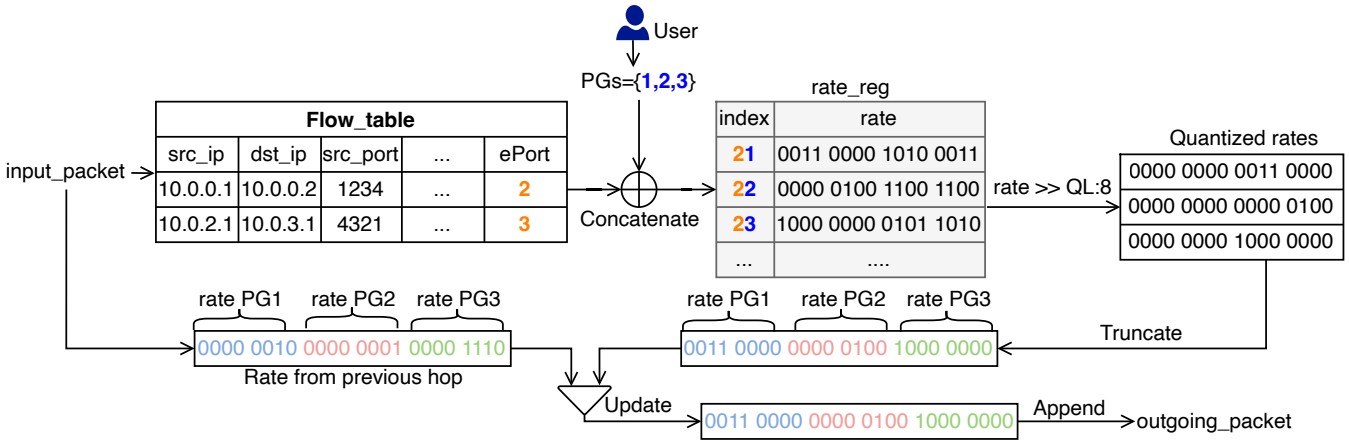


Figure 3: The internal of PrioMeter for per-PG data rate measurement: flow table, rate register, quantization, and truncation

the number of PGs in the network. Since the P4-enabled device accumulates the size of the packets in the `bytes_reg` register, we need to reset its value periodically over a specific measurement time window. This reset implies that PrioMeter should keep track of the timestamp of the incoming packets to sum the packet lengths.

We use the `timer_reg` register to account for timing since our measurements rely on high-resolution timestamps provided by programmable devices such as Tofino [1]. PrioMeter initialize this register with the timestamp of the first packet and resets its value to the latest timestamp when it reaches the threshold value.

Running example. In Figure 2, we provide a running example to show how PrioMeter performs per PG data rate measurement in the data plane. We assume that all the packets belong to the same PG for simplicity. At the beginning of measurement, we assume that the value of the timer_reg is zero and receive three packets, namely, A, C, and F during this time window, i.e., *window*#1. We also assume that four packets are received in the second time window, i.e., *window*#2.

Next, we explain how PrioMeter updates the values of the different registers. PrioMeter initializes the values of `bytes_reg` and `rate_reg` registers to zero and `timer_reg` to the timestamp of packet A. PrioMeter updates the values of `bytes_reg` and `rate_reg` registers when the timer threshold value is reached. It checks the size of packet A, i.e., 10 Bytes, and writes this value into the `bytes_reg` register. Since the threshold for the update is not reached, the values of `bytes_reg` and `rate_reg` registers remain unchanged. The same procedure happens for the new two consequent packets. After receiving packets C and F, the value of `bytes_reg` register changes to 30 and 60 Bytes, respectively. By arriving packet B, the time threshold to reset the values of `bytes_reg` and `timer_reg` and update the value of `rate_reg` registers are met. PrioMeter writes 60 as the measured rate into the `rate_reg` register. The same procedure happens for the next four consequent packets in time window#2.

Per-priority measurement of the PrioMeter adds the measured data rate information of each priority to the header of each packet. We use a fixed-width packet header to carry the rate information. PrioMeter checks the value of the header field and updates its value according to the per-priority aggregation function. In our architecture, the receiver extracts the measured data rate value.

To store the data rate of each PG, we need to design a feasible and efficient data plane data structure. The data structure should be able to keep the data rate of different priorities and ports of each switch. The straightforward approach is to use a two-dimensional array with a possible number of ports and priorities. However, this approach is not implementable in real hardware switches due to the limitations on the number of register accesses. We introduce an alternative design choice that we call *priority meter data structure*. Our approach concatenates the egress port number and the PG of the packet to find the index of the register to update the data rate. Considering a commodity network device with 64 ports and eight PGs in the network, PrioMeter requires 512 memory cells to keep track of the data rate of all PGs per port.

To aggregate the data rate of each PG, we need to collect a set of data rate values for PGs and append its measured value to the current packet. Assuming a 32-bit field value for keeping the data rate, we need to add $|PGs| \times 32$ bits overhead to each packet to do the measurement. This adds a significant amount of overhead to the packets resulting in reducing the performance of the network particularly in datacenters. Furthermore, it requires more bandwidth to carry the data. To overcome this challenge, we leverage *quantization* to encode the measured data rate and reduce the overhead of our estimation. After quantizing the data rate of each PG, we truncate each data rate and use a predefined offset to append them into the packet. PrioMeter performs the following three steps: (i) collect per-PG data rate, (ii) update, and (iii) decode. We now explain them in detail.

3.3 Collecting per-PG Data Rate

To collect the data rate estimation of all PGs when they are copied to *rate* register, PrioMeter employs two basic primitives: *quantization* and *truncation*. The first one aims at measuring the precise data rate information of each PG to coarse-grained bins representing a range of continuous data rates. The second primitive trims the number of bits that are needed to carry the data rate information of each PG. These two primitives reduce the overhead of carrying

the data rate information on the network. Figure 3 shows these operations. We now explain the details of each step.

3.3.1 Lookup. PrioMeter first matches the incoming packet with the flow table to lookup the associated egress port. The routing rules are proactively populated by the operator on the flow tables via the control plane. In Figure 3, for example, the rule ⟨10.0.0.1, 10.0.0.2, 1234⟩ specifies that the incoming packet should be forwarded via egress port 2. The egress port number of the incoming packets and the set of PG values are necessary to find the index of the flows in the `rate_reg`. The values of PGs in our example in Figure 3 are 1, 2, and 3. We extract the current value of the PG from the packet header.

PrioMeter concatenates the egress port number and value of each PG to create an index for each cell within various registers, such as `rate_reg`. Figure 3 shows that the index values for egress port 2 when the operator aims to measure the three PGs are 21, 22, and 23.

3.3.2 Quantization. The lookup step determines the egress port of the packet and what is the affected index in the `rate_reg` for data rate measurements. Herein, the first step is determining the correct bins for quantization to get the data rate. The bins for a specific PG indexed by *bins(index,rate)=rate(index)*$\gg QL$, where *index* is the same index obtained from the *lookup* step, QL is the quantization level, and $0 \leq QL \geq \log_2(c)$. Here, c is the maximum capacity of the link in bits. PrioMeter uses power-of-two bins to simplify the estimation to find the correct bin using the right shift in the data plane.

Figure 3 shows the data rate for each index in binary format. For example, the data rate value for index 21 is ⟨0011 0000 1010 0011⟩, with $QL = 8$ yields the quantization value for 48, i.e., ⟨0011 0000⟩, bps. In this example, we use 16 bits for the data rate measurement to simplify the presentation of our approach.

3.3.3 Truncation. After obtaining the quantized value in the second step, truncation trims the obtained value using bit slicing supported by the P4. It concatenates each value based on the offset specified by the number of PGs. Truncation is pivotal in mitigating the overhead introduced by PrioMeter when metering the data rate for each priority. For instance, in our example in Figure 3 with *QL=8*, PrioMeter receives the quantized value, i.e., ⟨0011 0000 1010 0011⟩, and takes the eight lower bits, i.e., ⟨0011 0000⟩, and appends them to the current packet header.

3.3.4 Concatenation. Finally, PrioMeter concatenates the truncated data rates and compares them with the received data rate from the previous hop, if any. We apply the per-priority aggregation to find the maximum measured data rate. To extract the current data rate values on the packet header, PrioMeter uses the same offset value after applying the bit slicing. Finally, it appends it to the packet header before sending it via the egress port. PrioMeter repeats these steps for every packet.

In our example in Figure 3, the current truncated value is ⟨0011 0000⟩, while the carried data rate is ⟨0000 0010⟩. Since the current truncated value is greater than the carried value on the packet header, PrioMeter appends ⟨0011 0000⟩ as the data rate for PG1.

3.4 Updating Per-PG Data Rate

The transmitted packets embedded with the data rate measurements reach the intermediate hops along the path before reaching the destination. PrioMeter updates the data rate values for each PG after checking them with their corresponding values in `rate_reg`. The step includes the following operations:

1. Data rate extraction. First, PrioMeter extracts truncated values by checking the header value of the packet and using the offset value of PGs. Like the truncation step, PrioMeter leverages the same offset values in this step.

2. Dequantization. After extracting the truncated data rate values, PrioMeter dequantizes them using the QL. For instance, applying $QL = 8$ in Figure 3 yields the actual data rate value for each PG. In our example, the dequantized value for the data rate of index 21 is 12,288 bits per second.

3. Update. If the obtained data rate value for a PG is higher than the current value, PrioMeter updates it and starts the quantization the same as collecting per-PG data rate. At the end of this step, the packet is sent through the designated egress port toward the destination.

3.5 Decoding Per-PG Data Rate

After collecting the data rate per PG by the PrioMeter, we can extract the measured data from the receivers. This step includes extracting the header values and *dequantizing* them as in the previous step. The received packets at the receiver have the data rate information of all PGs for each end-to-end path.

3.6 Theoretical Analysis of Quantization for Data Rate Measurement

By leveraging quantization for data rate measurement, we aim to efficiently encode continuous data rate values into discrete quantized levels using power-of-two numbers because of its support in P4 using bit shift operators. We now analyze its impact on accuracy.

3.6.1 Quantization Levels and Precision. Quantization involves partitioning the continuous range of data rate values of PGs into a finite set of discrete QLs. Assume that the number of QLs, denoted by $\mathcal{N}$, is determined by the number of bits used for quantization (β) in the second step of collecting per PG data rate. We can calculate $\mathcal{N}$ as follows.

$$\mathcal{N} = 2^{\beta} \tag{2}$$

The choice of $\mathcal{N}$ by the operator directly influences the precision of the quantized representation. A smaller β produces fewer QLs, leading to coarser quantization intervals and potentially lower approximation error. In contrast, larger β offers a fine-grained quantization and adds higher approximation error. These two approximation errors are due to the number of right shifts PrioMeter performs on the measured data rates to compress them.

3.6.2 Quantization Error. Using quantization introduces a non-negligible approximation error due to mapping continuous values

to discrete QLs. We define this error Q_e as the difference between the original data rate value (r_{pg}) and its quantized representation (r^q_{pg}):

$$Q_e = r_{pg} - r^q_{pg} \tag{3}$$

As β decreases, the quantization intervals become larger, and the value of Q_e increases. Conversely, increasing β reduces the size of quantization intervals and the value of Q_e. We note that using the sliding window for the data rate measurement may introduce additional estimation error, and we plan to investigate this as part of our future work.

3.6.3 Data Rate Compression Efficiency. Quantization enhances data rate compression efficiency by representing data rate values using fewer bits. Adding fewer bits to the packet header to perform the per PG estimation can reduce the negative effect of our measurements on the flow completion time and bandwidth usage. The number of compressed bits to store the results of quantization and truncation applied by PrioMeter can be calculated as follows:

$$\eta = |PGs| \times \beta, \tag{4}$$

where $|PGs|$ is the total number of priority groups applied to different traffic flows. However, data rate compression comes at the cost of potential quantization error. The trade-off between header savings and acceptable approximation error requires carefully selecting β based on the application's tolerance to quantization error.

4 PRELIMINARY RESULTS

We implement PrioMeter on P4 for the Behavioral Model v2 (BMv2) switch [7] to check its feasibility and NS3 for the large-scale evaluation. In this section, we report the results of the priority aggregation-based approach using NS3 simulations [2]. We do simulations using a dumbbell topology with two end-hosts connected to each switch, and all links are 100 Gbps. We report the measurement accuracy and the impact of different sliding windows when setting the quantization level to 12. By setting this QL value, PrioMeter does 12 right shifts on the measured data rate per PG, leading to at most 32 Kbps estimation error without considering the length of the measurement window. We generate traffic using the *webSearch* [4] benchmark workload for scenarios when the link load is 80%. The servers generate flows according to the Poisson process toward one of two random servers as in previous studies [5].

We generate traffic for different priority groups (PGs) based on the flow sizes. Table 1 reports how we classified them in our experiments based on the cumulative flow size distribution of *webSearch* workload [4].

4.1 Impact of Sliding Window Size

We first measure the impact of the sliding window size w on the rate of different PGs, see Figure 4. We first generate 51 flows with eight PG values in which their classification is based on their sizes (for details please see Table 1). We report the results of measurements for scenarios with 10, 50, and 100 microseconds sliding widows. We observe that PrioMeter achieves the best accurate results when the sliding window size is 50 microseconds. By checking the PGs of flows, we find the higher priority flows with lower sizes completed earlier than the lower ones, and PrioMeter reports zero rates when their corresponding flows finished. Figure 5 reports the results of measurements for different sliding windows when the QL=14. We can observe that increasing the quantization level when the sliding window is small reduces the accuracy of PrioMeter. Therefore, operators can set QL according to the capacity of their links.

Table 1: Traffic size classes based on flow size distribution of *webSearch* workload [4].

Class	Flow size range
PG 1	0-10 KB
PG 2	10 KB - 50 KB
PG 3	50 KB - 200 KB
PG 4	200 KB - 1 MB
PG 5	1 MB - 2 MB
PG 6	2 MB - 5 MB
PG 7	5 MB - 10 MB
PG 8	> 10 MB

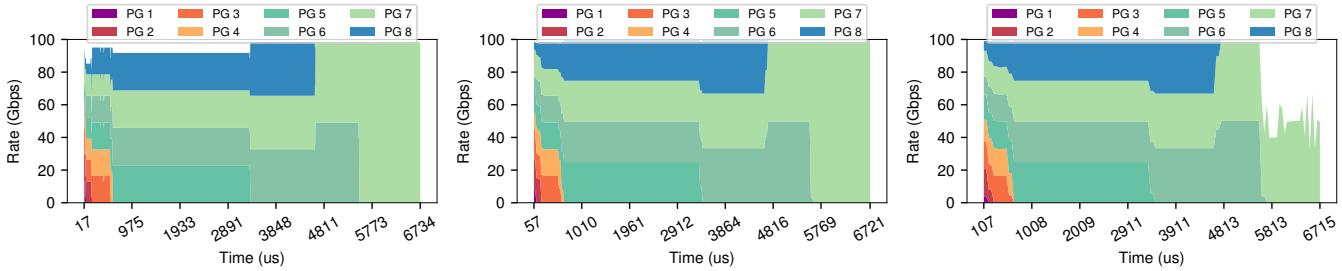
4.2 Comparison with Other Programmable Tools

Then, we report on the effect of reporting the data rate information of each priority group on the FCT of different workloads. We test the performance of PrioMeter on the same FatTree topology as HPCC [13] with 16 Core switches, 20 Agg switches, 20 ToRs, and 320 servers (16 per rack). Each server is connected to a ToR switch via a 100 Gbps NIC. The capacity of all interlinks among the Core, Agg, and ToR switches is 400 Gbps. In this topology, all the links have a 1-microsecond propagation latency, and the buffer size of each switch is 32 MB. We generate traffic using the flow size distribution of *webSearch* [4] and use the same setting as in the first experiment to assign PGs for various flows sizes. The servers generate flow according to the Poisson process toward random servers.

Figures 6(a) and 6(b) show that PrioMeter has a similar FCT slowdown compared with HPCC when the flow sizes are small, i.e., less than 256KB. However, HPCC adds more traffic overhead to the network when the flows have more packets to send since it adds the telemetry information from all hops to all packets. In contrast, PrioMeter adds a fixed amount of traffic overhead to packets, and the values of the overhead change at the end of the measurement windows. Notice that HPCC is not designed to report the data rate per PG. Assigning different PGs for the flows based on their size in a priori aims at better traffic scheduling, and PrioMeter adds less traffic overhead to each packet header when measuring the data rate at each hop. Moreover, the amount of overhead in HPCC depends on the number of hops since it collects extra INT telemetry information per hop, including timestamp, egress port utilization, and queue length. On the contrary, PrioMeter's overhead does not increase with the number of hops in the network, thus, it scales better in large networks.

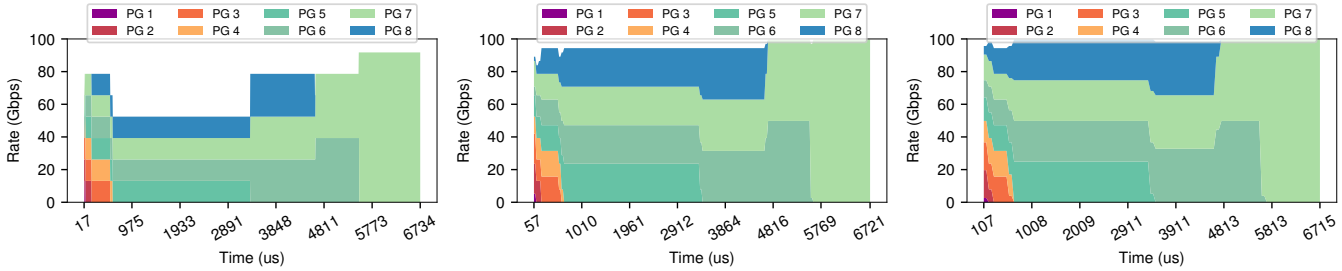
5 CONCLUSION

In this paper, we design, implement, and evaluate a new end-to-end per priority flow data rate estimation tool, PrioMeter, tailored



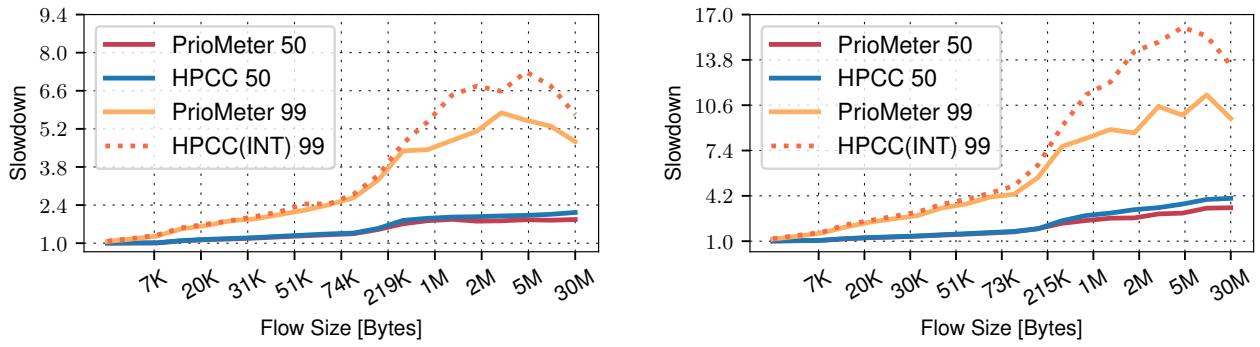
(a) w=10 us (b) w=50 us (c) w=100 us

Figure 4: The data rate measurement of PrioMeter for different sliding window lengths when QL=12.



(a) w=10 us (b) w=50 us (c) w=100 us

Figure 5: The data rate measurement of PrioMeter for different sliding window lengths when QL=14.



(a) Load=30% (b) Load=50%

Figure 6: The FCT slowdown for PrioMeter and HPCC on *webSearch* workload for 95- and 99-FCT percentiles. when the link loads are 30% and 50%.

for the programmable networks setting. With PrioMeter we include the information of the measured data rates of various priority groups to the packets to carry the data rate of each hop along the path, which is now readily available in P4 programmable switches. PrioMeter leverages two primitives, namely, quantization and truncation to achieve its goals. We implement PrioMeter in P4 and test it on BMv2 switches and study different aspects of PrioMeter in measuring the data rate of different priorities using NS3 simulations. We show that the performance of our tool is comparable with the state-of-the-art tools, yielding minimal overhead while estimating per priority flow data rate. We make our PrioMeter implementation publicly available. We plan to implement and test our system on Tofino switches as part of our future work.

ACKNOWLEDGEMENT

This research was supported in part by the European Union Horizon Europe research and innovation programme under grant agreement number 101092912 (project MLSysOps), by the German Federal Ministry of Education and Research under the grant BIFOLD23B, and by the European Research Council (ERC) under Starting Grant ResolutioNet (679158).

REFERENCES

[1] 2022. Intel Intelligent Fabric Processors. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html.

[2] 2022. NS3 Network Simulator. https://www.nsnam.org/.

[3] 2023. https://github.com/mostafaei/PrioMeter.

[4] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM*.

[5] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. 2023. Bolt: Sub-RTT Congestion Control for Ultra-Low Latency. In *Proceedings of USENIX NSDI*.

[6] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic In-Band Network Telemetry. In *Proceedings of the ACM SIGCOMM*.

[7] BMv2 simple_switch 2023. Behavioral Model v2. https://github.com/p4lang/behavioral-model.

[8] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95.

[9] Ningning Hu and Peter Steenkiste. 2003. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications* 21 (09 2003), 879 – 894.

[10] iperf3 2022. The ultimate speed test tool for TCP, UDP and SCTP. https://iperf.fr/iperf-doc.php.

[11] Manish Jain and Constantine Dovrolis. 2002. End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proceedings of the ACM SIGCOMM*.

[12] Nicolas Silveira Kagami, Roberto Irajá Tavares da Costa Filho, and Luciano Paschoal Gaspary. 2020. CAPEST: Offloading Network Capacity and Available Bandwidth Estimation to Programmable Data Planes. *IEEE Trans. Netw. Serv. Manag.* 17, 1 (2020), 175–189.

[13] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the ACM SIGCOMM*.

[14] Robert MacDavid, Xiaoqi Chen, and Jennifer Rexford. 2023. Scalable Real-Time Bandwidth Fairness in Switches. In *IEEE INFOCOM 2023*.

[15] William M. Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C. Snoeren, and George Porter. 2020. Expanding Across Time to Deliver Bandwidth Efficiency and Low Latency. In *Proceedings of USENIX NSDI*.

[16] NetPerf 2022. Netperf. https://github.com/HewlettPackard/netperf.

[17] Ravi Prasad, Constantine Dovrolis, Margaret Murray, and KC Claffy. 2003. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network* 17, 6 (2003), 27–35.

[18] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. 2003. pathchirp: Efficient available bandwidth estimation for network paths. In *PAM*.

[19] TTCP 2022. ttcp(1) - Linux man page. https://linux.die.net/man/1/ttcp.

[20] Hong Zhang, Junxue Zhang, Wei Bai, Kai Chen, and Mosharaf Chowdhury. 2017. Resilient Datacenter Load Balancing in the Wild. In *Proceedings of the ACM SIGCOMM*.