Fine-Grained Queue Measurement in the Data Plane

Xiaoqi Chen
Princeton University
xiaoqic@cs.princeton.edu

Shir Landau Feibish
Princeton University
sfeibish@cs.princeton.edu

Yaron Koral
AT&T Labs
yk216h@att.com

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

Ori Rottenstreich
Technion
or@cs.technion.ac.il

Steven A Monetti
AT&T Labs
sm1818@att.com

Tzuu-Yi Wang
AT&T Labs
tw503j@att.com

ABSTRACT

Short-lived surges in traffic can cause periods of high queue utilization, leading to packet loss and delay. To diagnose and alleviate performance problems, networks need support for real-time, fine-grained queue measurement. By identifying the flows that contribute significantly to queue build-up directly in the data plane, switches can make targeted decisions to mark, drop, or reroute these flows in real time. However, collecting fine-grained queue statistics is challenging even with modern programmable switch hardware, due to limited memory and processing resources in the data plane. We present ConQuest, a compact data structure that identifies the flows making a significant contribution to the queue. ConQuest operates entirely in the data plane, while working within the hardware constraints of programmable switches. Additionally, we show how to measure queues in *legacy* devices through link tapping and an off-path switch running ConQuest. Simulations show that ConQuest can identify contributing flows with 90% precision on a 1 ms timescale, using less than 65 KB of memory. Experiments with our Barefoot Tofino prototype show that ConQuest-enabled active queue management reduces flow-completion time.

CCS CONCEPTS

• **Networks** → **Data path algorithms**; **Network measurement**;

KEYWORDS

Network Monitoring, Queue Measurement, SDN, P4

ACM Reference Format:
Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. 2019. Fine-Grained Queue Measurement in the Data Plane. In *CoNEXT '19: International Conference On Emerging Networking Experiments And Technologies, December 9–12, 2019, Orlando, FL, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3359989.3365408

1 INTRODUCTION

In packet-switched networks, the queues that buffer packets awaiting transmission are fundamental components of the network. Much of the packet losses and delays that occur in the network are caused by backlogs in these queues. Yet, existing network devices offer surprisingly little visibility into the state of the queues, making it difficult to detect, diagnose, and fix performance problems.

In this paper, we introduce ConQuest, a measurement data structure that estimates the size of flows in a queue in real time, and identifies those flows making a significant contribution to queue occupancy. This is useful for a wide range of applications, from preventing congestion-related attacks to implementing active queue management (AQM) schemes. ConQuest performs measurements on small timescales, directly in the data plane of high-speed commodity switches.

Fine-grained, real-time queue monitoring is now possible with the emergence of commodity programmable switch hardware. To operate at 100 Gbps per port, PISA (Protocol Independent Switch Architecture) switches [9, 33] process packets in a pipeline of match-action tables and register arrays on both the ingress and egress ports, as shown in Figure 1. A match-action table can *match* (dark blue) on both packet header fields and metadata, such as the queuing metadata that is available in the egress pipeline. Subsequently, the *action* (purple) can be a forwarding or dropping decision, an update to header or metadata values, or an update to register memory (yellow). Once a packet is processed by the ingress pipeline, with its output port determined, the packet is placed in the appropriate outgoing queue, and later processed by the egress pipeline.

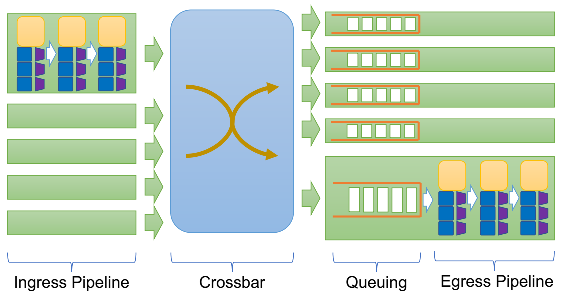


Figure 1: PISA switch with queues on egress ports.

PISA switches make it *possible* to monitor queues in the data plane, but they do not make it easy. To forward packets with high throughput and predictable, low latency, PISA switches impose several constraints. First of all, the switching pipeline has only a limited number of stages; the number of available actions and size of register memory per stage are also limited. Second, to simplify chip design and ensure low latency, each stage can only perform a handful of concurrent memory accesses to its own register array. Finally and most importantly, the ingress and egress pipeline cannot have access to the same register memory. This constraint is likely true for not just PISA but due to the fundamental mismatch between the aggregated speed of the link and the speed of memory. Since both pipelines are processing packets at high clock rate, concurrent access to the same memory could create a memory hazard.

Due to the concurrent memory access limitation, a data structure cannot be updated both when a packet enters the queue and when it departs. Instead, we can only analyze the queue in one place, when a packet's queuing metadata becomes available. Prior research shows how to work within the PISA constraints to perform fine-grained *logging* of packet bursts [20, 32, 36]. However, given all of these constraints, designing a data structure that can *analyze* the queue buildup entirely in the data plane—rather than collecting logs for offline analysis—remains an unsolved problem.

ConQuest is an efficient measurement primitive for identifying flows that contribute significantly to backlogged queues on a small timescale, *entirely* within the data plane, enabling immediate control actions. ConQuest does not provide a mechanism for congestion control. Instead, ConQuest's measurements allow online querying of the queue content which could enable new AQM and traffic management schemes. In designing ConQuest, we make the following contributions:

Designing an efficient data structure (§ 3). ConQuest identifies how much each packet's own flow contributes to queuing delay. Flows can be defined at various levels of granularity (e.g., five-tuple, source-destination pair, or destination address) depending on the purpose, such as detecting a single bursty TCP connection or an end-host or service receiving large bursts of traffic. To process each packet only once, ConQuest maintains multiple compact "snapshots" of the queue occupants over time; each packet updates one snapshot and queries multiple past snapshots.

Realizing the data structure on hardware switches (§ 4). Each PISA switch imposes certain limitations on the number of pipeline stages, and the number of registers and actions available in each stage. Given these target-specific constraints, we generate a P4 [33] program that implements our data structure for that target. We use a Barefoot Tofino switch [34] to demonstrate a real-world implementation of queue measurement and management in the data plane.

Quantifying the benefits of queue measurement in the data plane (§ 5). Simulations with packet traces allow us to characterize how the number and size of snapshots affect measurement accuracy. ConQuest can achieve over 90% precision and recall using less than 65 KB of memory. We build a flow-based active queue management prototype on Tofino and run closed-loop experiments to show that ConQuest can help improve end-to-end performance.

Off-path monitoring of queuing in legacy routers (§ 6). Most legacy routers only report coarse-grained queue statistics on a large timescale. We introduce an off-path monitoring technique that taps multiple links of a legacy device, and feeds the data into a version of ConQuest extended to match the ingress and egress observations of the same packet. Fine-grained monitoring of legacy routers enables network operators to troubleshoot performance problems in their network. We use our prototype to analyze queuing in a Cisco CRS router and verify ConQuest's accuracy. We also deployed ConQuest in a campus network and successfully diagnosed queuing anomalies in the border router.

The paper ends with a comparison to related work (§ 7) and our conclusions and future research directions (§ 8).

2 QUEUE MEASUREMENT USE CASES

While networks typically rely on end-hosts to perform congestion control, fine-grained queue measurements at *switches* are still critical for a wide range of purposes, including:

Stopping congestion-related attacks. In a Shrew attack [21], a few bursty flows (each sent every few seconds, for a short duration) cause a large transient backlog in the queue. Quickly identifying the queue buildup, and the contributing flows, enables rapid mitigation of these attacks.

Avoiding conflicting workloads. Interactions across multiple connections, such as TCP Incast [3, 11], can cause sudden queue buildup, leading to high tail latency for big data applications. Identifying the responsible applications enables better scheduling, load balancing, and VM placement decisions in data centers.

Optimizing switch configurations. Queuing parameters, such as weights in active queue management (AQM) schemes like W-RED [27], are notoriously difficult to tune [14]. With a fine-grained understanding of queuing dynamics, network operators can better configure these parameters to the prevailing workload.

Deploying new AQM schemes. While congestion control has long been an area of innovation, deploying new AQM schemes is challenging due to a lack of fine-grained queue measurement support in switches. A data-plane queue measurement primitive would provide the metrics necessary for more sophisticated AQM schemes.

Debugging switch implementations. Queue management in high-speed switches is a complex mechanism, with flow control between multiple queues on different ports. Implementation mistakes by equipment vendors can lead to counter-intuitive phenomena, like high packet loss and delay during periods of low link utilization. These bugs are difficult and time-consuming to detect, let alone diagnose and fix, without better visibility into queuing dynamics.

Using switches with shallow buffers. Cheaper switches with small buffers are sufficient for many networks [4, 35]. Finding a way to monitor queues of legacy routers can help network operators decide whether they can adopt shallow-buffer switches without compromising performance. In addition, a data-plane queue-monitoring primitive in new commodity switches can help manage the limited buffer resources to run the network at high utilization.

3 CONQUEST DATA STRUCTURE

ConQuest is a measurement data structure that operates in *real time* (to detect and mitigate even short-lived queue buildup as it forms), at a *fine granularity* (to pin-point individual contributing flows), and with *high accuracy* (to make good decisions). Meeting these requirements is not easy. A 100 Gbps link sends new packets every few nanoseconds, and a transient congestion event may last less than a hundred microseconds [7, 38]. To analyze a queue on a small

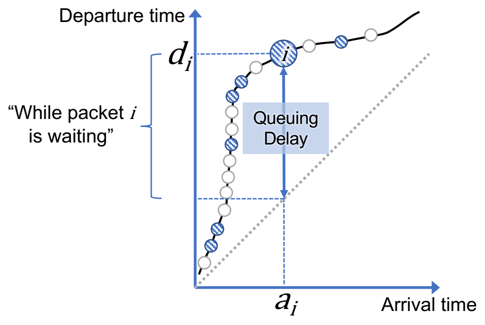


Figure 2: Packet departure time (d_i) vs. arrival time (a_i) in a queue. While packet i was queued, three (shaded) packets of the associated flow f_i departed.

timescale, we cannot rely on packet sampling or coarse-grained statistics such as queue length, as fine-grained information about transient congestion events would be lost with the high sampling rates (as low as 1 in every 30,000 packets [30]) in today's networks. Yet, processing every packet in software would not scale to high link speeds. Instead, our data structure must operate at line rate, within the data plane.

As discussed earlier, our data structure design is subject to a fundamental architecture limitation: we cannot concurrently perform updates at both ends of the queue, meaning, ConQuest cannot insert a packet as it enters the queue and later remove it when it departs. This encouraged us to design a snapshot-based data structure that passively *expires* groups of packets instead of actively deleting old packets. To estimate flow-level queue occupancy in real time, ConQuest combines results across multiple snapshots, and cleans and reuses expired snapshots in the background.

3.1 Contributing Flows in a Queue

For a constant-rate link serving a single FIFO queue, a packet's queuing delay corresponds directly to the length of the queue when it arrives. ConQuest identifies the flows that consume a large portion of the queue and are, therefore, significant contributors to the backlog. Tracking these flows would *seem* to require per-flow counters, updates to the counters on both packet arrival and departure, and identifying the largest counters at any given time. Realizing such a data structure in the data plane is inherently difficult, due to the constraints outlined in Section 1. Fortunately, we do not need to estimate the contribution of all flows all of the time, just *some* flows (i.e., the most significant contributors) *some* of the time (i.e., when we see a packet of that flow and queuing is long).

Querying a flow's own contribution to the queue: For the switch to take corrective action on the flows causing the backlog, we need only identify the contribution of the *current* packet's flow to the queue. More precisely, for each packet, we ask: *while this packet was queued, what fraction of the packets (or bytes) transmitted over the link belonged to its own flow?* As shown in Figure 2, for a packet i with flow ID f_i arriving at time a_i and departing at time d_i, all packets j with departure time $d_j \in [a_i, d_i)$ departed while packet i was waiting in the queue. Some of these packets belong to the same flow as i (i.e., $f_j = f_i$, shaded blue). As an example, in Figure 2, packet i was the tenth packet in the queue when it arrived, and three of those packets were from flow f_i.

Each egress pipeline witnesses packets leaving the queue as a stream of (f_i, a_i, d_i) tuples, where flow ID f_i is determined from packet headers and timestamps a_i and d_i are queuing metadata, which are available in the data plane after the packet leaves the queue. We define a queue to be congested when the queuing delay of the packets reaches a threshold of τ. When congestion occurs, ConQuest aims to identify the *contributing flows*, whose packets occupy at least an α fraction of the queue. Or, more formally:

Definition 3.1 (Contributing flow). Given a FIFO queue with a congestion threshold τ and a contribution threshold α, when packet i is departing with flow ID f_i and arrival/departure timestamps a_i, d_i, if $(d_i - a_i) \geq \tau$, *and:*

$$\frac{\left|\{j \mid (a_i \leq d_j < d_i) \ \ \& \ \ (f_j = f_i)\}\right|}{\left|\{j \mid a_i \leq d_j < d_i\}\right|} \geq \alpha$$

then f_i is currently a *contributing flow.*

For ease of exposition, we assume that all packets have unit size; however, it is straightforward to extend the definitions to consider packet length.

Accuracy when it matters: Hence, to understand queue backlog, ConQuest needs to report accurate estimates (i) only for the *contributing* flows, rather than the many less significant flows, and (ii) only when the queuing delay is high. This allows ConQuest to use approximation techniques to work within the constraints imposed by PISA switches.

3.2 Traffic Snapshots for Bulk Deletion

To determine if a packet is part of a contributing flow, ConQuest maintains information about past packet departures. When packet i departs the queue, ConQuest queries packets from the *past* based on the time range $[a_i, d_i)$, and also inserts the current packet's flow ID and departure timestamp (f_i, d_i) into the data structure to support *future* queries.

The main challenge of performing these operations on PISA switches is to accurately *delete* information about packets whose departure timestamp has become too old to be relevant to any future packet's query. Since packets do not actively delete themselves, we group packets into fixed time-window snapshots of length T based on their departure time, allowing us to passively expire a window of past packets in bulk. Let us choose T=3 for demonstration: in Figure 3(a), the rightmost packet (shaded blue) with departure time 0 goes into snapshot $\lfloor 0/T \rfloor = 0$. The next packet from flow A (also in blue) departs later at time 4 (as shown in Figure 3(c)), thus falling in snapshot $\lfloor 4/T \rfloor = 1$.

For each snapshot, we count the total number of packets for each flow; for example, snapshot #0 in Figure 3(a) has one packet for flow A and two packets for flow C. Afterwards, we can query this snapshot to obtain the sizes of flows during this time window. Using snapshots, we can implicitly expire old packets in bulk from the system by no longer querying the oldest snapshot. We can ignore expired snapshots, or better yet, *recycle* them (illustrated in Figure 3(c)) as discussed in more detail in Section 3.4.

If the number of flows is limited and known beforehand, each snapshot could consist of simple per-flow counters. For a network

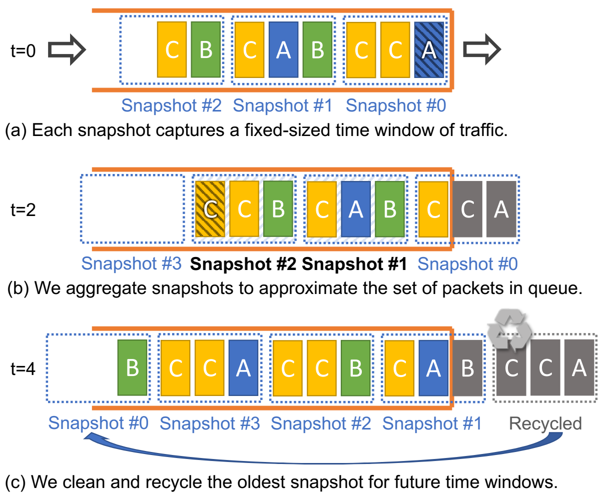

t=0
C B C A B C C A
Snapshot #2 Snapshot #1 Snapshot #0
(a) Each snapshot captures a fixed-sized time window of traffic.
t=2
C C B C A B C C A
Snapshot #3 Snapshot #2 Snapshot #1 Snapshot #0
(b) We aggregate snapshots to approximate the set of packets in queue.
t=4
B C C A C C B C A B C C A
Snapshot #0 Snapshot #3 Snapshot #2 Snapshot #1 Recycled
(c) We clean and recycle the oldest snapshot for future time windows.

Figure 3: Time-window snapshots on a queue.

with a large number of flows, per-flow counters are not feasible. While solutions such as Counter Braids[25] and FlowRadar [23] can record *precise* per-flow counts, they require offline decoding and thus cannot be queried from within the data plane. However, since we care about the *large* contributors to the queue, any *approximate* data structure that supports inserting or incrementing counts and querying flow sizes with reasonable accuracy can achieve our purpose; in our prototype, we use the Count-Min Sketch (CMS) [15] due to its ease of implementation in the data plane. The CMS can estimate flow sizes with a possible overestimation error due to hash collisions; the error bound depends on the selected size of the structure. Note that incrementing the CMS may be easily modified to estimate flow sizes based on either packet count or byte count.

Hence, ConQuest makes two kinds of approximations: (i) when we divide traffic into time windows, the query for a particular time range $[a_i, d_i)$ will be rounded into a query to an approximated range and (ii) the use of sketches can lead to overestimates in the flow counts in each window. We evaluate the effect of both types of error in Section 5.1.

3.3 Aggregating over Multiple Snapshots

Ideally, to decide if packet i belongs to a contributing flow, we would compute f_i's flow size within the departure time range $[a_i, d_i)$. ConQuest computes an approximate answer by looking at a number of recent time windows that are contained in $[a_i, d_i)$, namely from snapshot $\lceil \frac{a_i}{T} \rceil$ to snapshot $\lfloor \frac{d_i}{T} \rfloor - 1$ (we round towards the more conservative side, which also uses fewer snapshots). Thus, by aggregating the flow size of f_i in the corresponding snapshots, we know approximately how many packets from f_i departed during $[a_i, d_i)$. Since we can only aggregate an integer number of snapshots, our estimate of the queue's content will differ from the actual queue's head and tail, with "rounding error" no more than T on both sides. When the queuing delay $(d_i - a_i)$ is much larger than T, i.e., when the queue is backlogged and, therefore, we are interested in measuring, we have smaller relative error. Aggregating over multiple snapshots allows us to estimate longer queue with more snapshots and shorter queue with fewer snapshots; using only a single snapshot would result in always analyzing a fixed time window, which is less accurate given the varying queue length.

As a concrete example, in Figure 3(b), the leftmost packet (shaded yellow) from flow C arrived at a_i=2. This packet will ultimately depart the queue at d_i=8, assuming one packet departs the queue in each time unit. Once we know a_i and d_i in the egress pipeline, the packets of interest are those that departed in the time range $[2, 8)$, i.e., the seven packets shown *inside* the queue in Figure 3(b) (other than i itself); out of these packets, there are three packets from flow C (yellow packets). Snapshot #1 recorded one packet for flow C, while Snapshot #2 recorded two packets. By aggregating the two shaded snapshots, #1 and #2, we can get an approximate value 3, i.e., there are around three packets from flow C among the seven packets that departed between time $[2, 8)$.

Besides simple summation, we may also aggregate snapshots differently to compute other metrics in the data plane. This creates more applications for snapshots beyond analyzing congestion. For example, we can detect rapid changes in flow throughput in the data plane, by computing the difference between the flow sizes reported by the two most recent snapshots. This technique would help network operators locate flows which rapidly ramp-up without obeying congestion control. Furthermore, by operating only on packet arrival and departure times, ConQuest can analyze congestion under a range of queuing disciplines. In this paper, we mainly focus our discussion on a link with a single FIFO queue. The extension to more general queuing disciplines is relatively straightforward, and we leave the technical details to Section 3.6.

3.4 Cleaning & Reusing Expired Snapshot

ConQuest only needs a constant number of snapshots to analyze a FIFO queue of bounded length served by a constant-rate link. For example, a 20 Mb queue served by a 10 Gbps link would have a maximum queuing delay $\max(d_i - a_i) = 2$ ms. If each snapshot covers a time window of length $T = 1$ ms, ConQuest needs to read from at most two past snapshots. Namely, we can choose time window T based on the total number of snapshots h, such that aggregating all snapshot time windows would approximately cover the entire queue. When a snapshot is no longer useful, we can recycle the snapshot for recording future traffic, as shown in Figure 3(c). Since snapshots are rotated on a very small timescale, we cannot rely on the control plane to clean expired snapshots in a timely manner; there is also no straightforward way to batch clean the register memory in data plane. Instead, we clean expired snapshots gradually, one entry at a time.

More generally, ConQuest maintains $\lceil \frac{\max(d_i - a_i)}{T} \rceil$ snapshots for *reading* (i.e., queries), one for *writing* (i.e., for inserting new packets), and one for *cleaning* (i.e., recycling), for a total of $h = \lceil \frac{\max(d_i - a_i)}{T} \rceil + 2$ snapshots. As illustrated in Figure 4, the roles of snapshots rotate every T seconds, synchronized with the progress of the time window. Each packet i traverses all h stages, indexing the Count-Min Sketch with its own flow ID f_i in the reading and writing stages, and indexing with a global index for clearing part of the CMS in the cleaning stage. In summary, in handling packet i from flow f_i, ConQuest performs the following operations based on its arrival and departure timestamps a_i, d_i:

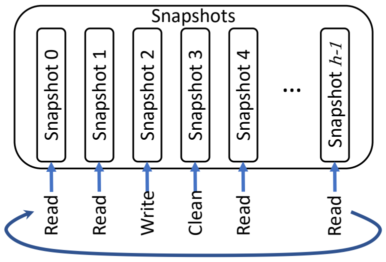


Figure 4: Round-Robin between h Snapshots. In any given time window, ConQuest writes into one snapshot, reads many, and cleans one for the next time window. Snapshot roles are rotated every time interval T.

	Definition
a_i, d_i	Arrival and departure times of packet i
f_i	Flow identifier for packet i
w_{f_i}	Weight (size) of flow f_i across packets departed during $[a_i, d_i)$
W	Total weight (size) of all flows inserted into snapshots
h	Number of snapshots
R, C	Number of rows & columns in Count-Min Sketch
α	Threshold for identifying a contributing flow
τ	Queuing delay threshold
S	Number of pipeline stages available in switch
M	Number of concurrent register memory accesses per stage supported by switch

Table 1: Summary of notation

- *Write:* Increment the size of flow f_i in the CMS associated with snapshot $\lfloor \frac{d_i}{T} \rfloor$ mod h.
- *Read:* Accumulate the estimated size of flow f_i in snapshots $(\lfloor \frac{d_i}{T} \rfloor - 1)$ mod h, $(\lfloor \frac{d_i}{T} \rfloor - 2)$ mod $h, \ldots, \lceil \frac{a_i}{T} \rceil$ mod h. The number of aggregated snapshots varies, and depends on the time the packet spent in the queue.
- *Clean:* Zero an entry in snapshot $(\lfloor \frac{d_i}{T} \rfloor + 1)$ mod h. For a CMS with C columns, we maintain a global packet counter cnt, and write zero to the (cnt mod C)-th item in each row.

Using the cleaning technique described above, the CMS is cleaned after C packets—we choose C and T such that there are at least C packet departures in one snapshot time window of length T, say, at 10% link utilization. If this is not the case, i.e., the link is very underutilized such that the number of packets per time window T is smaller than C, the last packet (that departs a now-empty queue) can trigger a report to the control software to clean the snapshot; note that this software can run at a timescale relative to T, which is significantly slower than the timescale of individual packets. Alternatively, if the target switch supports packet generation (such as on the Barefoot Tofino Wedge-100 switch [34]), when the link is underutilized the data plane can generate the additional packets needed for cleaning the snapshot and filter them before the end of the egress pipeline.

3.5 Error Analysis

We now analyze the worst-case estimation error for the ConQuest data structure due to hash collisions, and show that when using h Count-Min Sketches each with R columns and C rows, it achieves $\varepsilon = e/R$ additive error with failure probability $\delta = (h-2)e^{-C}$.

First of all, each snapshot Count-Min Sketch [15] provides $\varepsilon = e/R$ additive error with $\delta_{CMS} = e^{-C}$ failure rate, which means with $(1-\delta)$ probability a query with ground truth flow size w will return an estimate $\hat{w}$ satisfying $w \le \hat{w} \le w + \varepsilon W_{CMS}$, with W_{CMS} being the total size of all inserted flows into this CMS.

ConQuest reads from at most $h-2$ snapshots to aggregate flow size estimates. Since each read has failure probability δ_{CMS}, we can use union bound to bound the probability of having any failure as $(h-2)\delta_{CMS}$. Therefore, the aggregate read's failure probability is $\delta = (h-2)e^{-C}$. When the aggregate read succeeded, the read error produced by each CMS is at most εW_{CMS_j}, and the total additive error for the output is bounded by $\sum_{j=1}^{h-2} \varepsilon \cdot W_{CMS_j} = \varepsilon \cdot W$, where W is the total size of all flows inserted into all $(h-2)$ snapshots. Thus, we show that ConQuest has additive error bound $\varepsilon = \frac{e}{R}$, i.e., returns estimated flow size $\hat{w_f}$ within $w_f \le \hat{w_f} \le w_f + \frac{e}{R}W$, with failure probability at most $(h-2)e^{-C}$.

Plugging in the parameters from our hardware prototype ($h = 4$, C=2,R=2048), we have worst-case additive error rate ε=0.0013 with maximum failure probability δ=0.27. We also analyze this error empirically in Appendix B.

3.6 Non-FIFO Queuing Disciplines

So far, we assumed that each link serves a single FIFO queue. In practice, links often use non-FIFO queuing, such as when an outgoing link has multiple FIFO queues (serviced by a scheduler), or even more exotic queuing disciplines. Here we describe some potential future works for utilizing ConQuest for analyzing queuing and congestion in general queuing disciplines.

Contributing flows within a traffic class. Under multiple traffic classes, a link may have one FIFO queue per class, as well as a scheduler (e.g., strict priority or weighted fair queuing). Since ConQuest considers only the packet arrival and departure times, the question "while i was waiting, what fraction of the packets transmitted over the link belonged to its own flow f_i" from Section 3.1 is still germane. The answer is useful to assess how much packet i's flow contributes to queuing for its own traffic class, and act on the current packet accordingly. However, unlike the case of single FIFO queue, the *maximum* queuing delay can be large (and, in the worst case, unbounded), under heavy load in higher-priority traffic classes. Instead, we can specify a maximum history to maintain, and answer queries about contributing flows relative to traffic departing during that bounded period.

Contributing flows across all traffic. More generally, high queuing delays for low-priority traffic may stem almost entirely from other, higher-priority flows that receive fast service (i.e., $d_i - a_i$ is small). By querying on the narrow range $[a_i, d_i)$, ConQuest would *not* realize that packets of flow f_i are adversely affecting other (lower-priority) flows. The query for packet i would report that few, if any, packets of flow f_i were transmitted while packet i was waiting! To analyze contributing flows *across* traffic classes (or across groups of flows with FIFO scheduling), we can slightly modify the definition of a *contributing flow* to enable these significant flows with small delay to recognize the harm they do to other traffic. In particular, ConQuest can maintain an additional register to store the maximum delay ($MaxDelay$) experienced by packets that recently left the queue, and perform a query for packet i that considers a larger time range $[d_i - MaxDelay, d_i)$. The value of $MaxDelay$ can decay gradually over time, when queuing delays are low.

4 P4 HARDWARE SWITCH PROTOTYPE

We implemented a prototype of ConQuest in P4 on a Barefoot Tofino Wedge-100 switch. We first show how to map ConQuest to different PISA targets and how we automatically generate target-specific P4 code. Then, we describe and implement some of the possible control actions the switch can take based on ConQuest measurements.

4.1 Mapping ConQuest to PISA Hardware

Although P4 is a target-independent programming language, different hardware targets may vary significantly from one another in characteristics like the number of pipeline stages, size of memory, number of concurrent actions in each stage, etc. Therefore, even though we designed ConQuest to fit within the PISA processing model, we still need to configure its parameters to fit within individual PISA hardware target's memory and processing capacities.

Automated generation of target-specific P4 code. To facilitate the use of our code on different targets, instead of writing a P4 program, we write a parameterized program that can be instantiated with a range of parameter values. Namely, we implement ConQuest in P4 with inline C-style macros. Once we specify parameters h, R, and C, a compiler automatically generates the expanded P4 code that fits the constraints of the specific hardware target.

The parameterized P4 program is roughly 900 lines, 60% of which are boilerplate code supporting packet parsing, hashing etc, and 40% are ConQuest's snapshot logic. The program first parses the IP and TCP/UDP headers to obtain the 5-tuple as the packet's flow ID. Then, it computes hash functions over the flow ID for reading or writing the Count-Min Sketches. The header parsing and hashing steps are programmable, and can change to use other flow ID definitions (e.g., source-destination pair, destination IP, etc.).

Mapping the logical structures to physical hardware. We now discuss how to map the *logical* structure presented in Section 3 to the pipeline stages in a hardware target. We assume ConQuest is only allowed to use S pipeline stages to manipulate snapshots, constrained by the capacity of the hardware target, and further limited by the other duties the switch must perform, such as packet forwarding. Additionally, we denote M as the maximum number of concurrent register accesses the target can support in each stage.

Assume a ConQuest implementation with h snapshots, and with R rows and C columns in each snapshot's CMS. Each CMS uses R register arrays, and reads/updates one entry per array for each packet. We therefore need $h \times R$ register accesses per packet in total in the worst case, which implies a necessity for at least $\lceil \frac{h \times R}{M} \rceil$ stages for memory access. Since each snapshot operates independently, at each stage we can "stack" multiple rows of different snapshots, and perform the read, write, or clean operations concurrently.

After reading the snapshots, we need another $\lceil \log_2(h-2) \rceil$ stages to sum the counts from all snapshots. We require the total number of stages used to manipulate snapshots not exceed the available stages: $\lceil \frac{h \times R}{M} \rceil + \lceil \log_2(h-2) \rceil \leq S$. Additionally, we need a small constant number of stages for pre-processing, such as computing the read/write/clean roles for snapshots and the memory addresses to use in the CMS.

ConQuest needs to fit into the hardware resource constraints of programmable switches while sharing resources with other switch functionality; under these constraints, we would choose the largest possible values of h, R, and C to achieve optimal accuracy. We further discuss the effect of each parameter on accuracy in Section 5.1. Furthermore, since arbitrary division is not supported on PISA hardware targets, we implement division and floor operations using bit right-shift, and implement modulus using bit slicing, which are explicitly defined in P4 specification [33]. Consequently, we choose both T and h to be integer powers of 2.

4.2 Actions on the Contributing Flows

In this section, we discuss how ConQuest allows the switch data plane to take action on packets based on a flow's contribution to queue backlog. As we discuss later in Section 5.2, we have implemented flow-based ECN marking and dropping in our ConQuest prototype to prevent contributing flows from further deteriorating congestion. We discuss these, as well as other potential solutions.

Marking/dropping based on flow weight. When the queue builds up, the data plane can mark the Early Congestion Notification (ECN) bit of the packets; if the queue grows even longer, the switch can go further and start dropping packets. In conventional Random Early Detection (RED) [16] schemes, the packets from different flows are simply dropped (or marked) with the same probability depending on average queue utilization. ConQuest enables the switch to decide actions on packet i from flow f_i, based on the current size of flow f_i in the queue when i arrived, denoted as *flow weight* w_{f_i}. As a basic example, given a threshold w_T, we mark packet from flow f only if $w_f \geq w_T$. This will throttle the heaviest flow in the queue, while leaving small flows intact. We can also probabilistically drop (or mark) the packet with probability $\Pr[\text{ECN}] \propto max(w_f - w_T, 0)$, or with other more sophisticated probability functions such as $\Pr[\text{ECN}] \propto max(w_f - w_T, 0)^2$, inspired by CHOKe [28]. In this way, ConQuest enables fast prototyping of active queue management algorithms that target contributing flows by using probabilistic dropping, based on the individual flow's size in the queue.

Our P4 prototype of ConQuest supports dynamically specifying threshold w_T at run time to achieve threshold-based ECN marking or dropping for contributing flows. We demonstrate the effectiveness of this flow-based ECN approach in Section 5.2. The prototype also supports piecewise constant approximation of any ECN marking (or dropping) probability function based on w_f.

Act on future packets. Upon identifying a contributing flow, the switch can feed its ID from the egress pipeline back to the ingress pipeline using packet recirculation. The ingress pipeline may then prevent this flow from exacerbating the imminent queue buildup, by re-routing, rate-limiting, or dropping its packets.

Report flow IDs. Transient congestion is sometimes not caused by individual contributing flows. In some cases, we can identify the cause of the congestion by defining flows at a coarser level of granularity. For example, TCP Incast [11] is caused by many sources sending packets to the same destination simultaneously, and can be

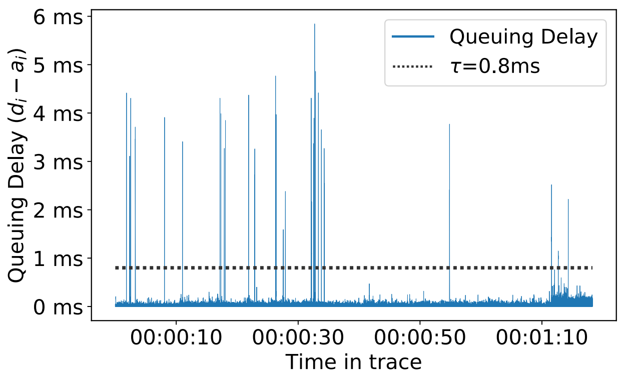


Figure 5: Simulated queue buildup on the UW Trace shows low average utilization with occasional bursts.

accurately captured by defining flows by destination IP address. In other cases, ConQuest can report packets from contributing flows to a software collector for further analysis, such as aggregating the reports to detect hierarchical heavy hitters or other groupings of flows belonging to a single distributed application (e.g., coflows [12, 13]). We leave these extensions as future work.

5 EXPERIMENTAL RESULTS

We evaluate ConQuest using two different setups. We use multi-factor simulation experiments to test the accuracy of ConQuest under different parameter settings. We do so by comparing ConQuest's output to the ground truth found in the simulation. Subsequently, we verify ConQuest's effectiveness in detecting and acting on the flows contributing to a backlogged queue, by evaluating our ConQuest prototype in a real-world testbed. We show that using measurements from ConQuest, the switch can throttle bursty flows to reduce median flow completion time.

For consistency across our experiments, we match the link rates of the legacy equipment and use an egress line rate of 10 Gbps in all of our experiments, including the tapping setup in Section 6.3. Selecting this link rate affects the timescale of a backlogged queue. On a 10 Gbps link, using a 40 Mbit buffer space (typical in commodity switches) leads to a maximum delay of 4 milliseconds, and a snapshot time window of T=1 ms using 4 snapshots; at 100 Gbps line rate, we would have 400 microseconds maximum delay and ConQuest would run with T=100μs.

5.1 Multi-Factor Simulation Experiments

Simulation experiments allows us to freely tune all parameters of ConQuest that practical hardware may not permit, and gives us full detail about the queuing dynamics at any given time. Therefore, we use simulations to evaluate ConQuest's accuracy while changing its parameters.

Dataset and implementation. To simulate queuing delay, we utilize the publicly available University of Wisconsin Data Center Measurement trace *UNI1* (UW trace) [7], by feeding the trace through a single FIFO queue with constant 10 Gbps drain rate and unlimited queuing buffer. We use the UW data-center trace in our experiment as it is the only public trace we are aware of that exhibits significant burstiness for simulating queue buildup, while other public traces such as CAIDA are less bursty. Since the original trace is published when links are predominantly 1 Gbps or 100 Mbps, and the trace has an average throughput of only 25.3 Mbps, we replay the trace 50x faster to reach 7.5% average link utilization at 10 Gbps. As seen in Figure 5 the queue length exhibits a bursty pattern over time. Similar pattern arises when we calibrate the trace to 3.75% or 15% utilization (replay 25x or 100x). The maximum queue utilization during the replay is around 8 MB (6.4 ms at 10 Gbps).

We simulate the queuing delay and ConQuest snapshots using Python. When a packet i experienced queuing delay ($d_i - a_i$) greater than τ=0.8 ms (about 1/8 of maximum queue depth observed), ConQuest reads past snapshots and reports an estimated flow size in the queue w_{f_i} for flow f_i when i entered the queue. Flow f_i is flagged as a contributing flow if w_{f_i} exceeds α fraction of the queue length. Note that for FIFO queues, "packets in queue at time a_i" is equivalent to "all packets departed during $[a_i, d_i)$". We also use simulation data to compute the ground truth contributing flows based on actual flow sizes. We first show results for α=1% as a representative threshold, and later show that ConQuest is robust for various choices of α.

We note that ConQuest is answering an imbalanced binary classification problem, as the packets belonging to contributing flows are not half of all packets queried. Therefore, we use Precision and Recall analysis to precisely describe ConQuest's accuracy. *Precision* is defined as the number of packets correctly identified by ConQuest as part of a contributing flow divided by the number of all packets reported by ConQuest. *Recall* is defined as the number of packets correctly identified by ConQuest as part of a contributing flow divided by the ground truth number of packets belonging to contributing flows. As a standard metric for evaluating a binary classifier, Precision and Recall capture how ConQuest trades false positives for false negatives and achieves balanced accuracy.

We define a flow based on the standard 5-tuple (source and destination IP address, protocol, and source and destination port). The UW trace has around 550,000 distinct flows in total. In our queuing simulation, when the queue is congested, there are on average 63.6 distinct flows in the queue (with 130 flows at 95%-percentile and 200 flows at 99.9%-percentile), out of which there are an average of 3.7 contributing flows (for α=1%).

There are two primary design choices for ConQuest, the snapshot data structure's memory size and the snapshot time window size. Using more memory to construct larger Count-Min Sketch (CMS) data structures reduces collisions and improves accuracy. Using a smaller time window T provides better granularity when approximating the range $[a_i, d_i)$ by lowering the rounding error, at the cost of using more pipeline stages. We evaluate the effect of both on accuracy.

Effect of limited per-snapshot memory. We first evaluate the memory needed to achieve adequate accuracy. For each snapshot, we use a CMS with R=2 rows and vary the number of columns C. When C is small (hence using less memory), CMS suffers from hash collisions and over-estimates the size of flows, reporting more false positives and lowering Precision (but Recall does not change since CMS would not underestimate flow size). Figure 6 shows the

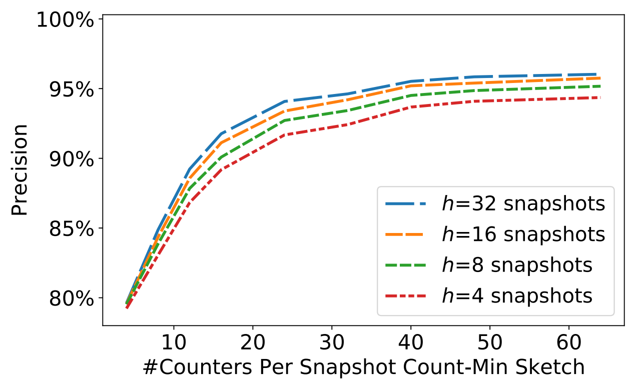


Figure 6: Precision vs. snapshot data structure size. Using 24-32 counters per CMS is adequate.

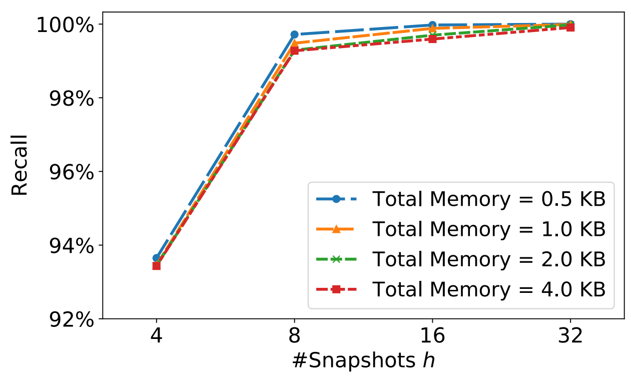


Figure 7: Recall vs. number of snapshots. Using 8 snapshots gives sufficiently high Recall.

effect of varying the total number of counters in the CMS on Precision. The Precision plateaus when there are $(R \cdot C)$=32 counters per CMS with diminishing returns for additional counters. This aligns with our observation that there are only tens of active flows in the queue during congestion and only very few heavy flows, hence even a small CMS can already distinguish heavy flows from small flows. With enough counters in the CMS, there is practically no hash collision. We note that Recall is influenced more by the number of snapshots and not by CMS size, as shown later in Figure 7. In real-world deployments, we should use a larger CMS with more counters if there are more active flows in the queue during congestion. We expect ConQuest to achieve nearly 100% Precision whenever the number of counters in CMS is approximately the number of active flows in queue.

Effect of snapshot time window size. Next, we evaluate the effect of snapshot window granularity on accuracy. Increasing the number of snapshots h (therefore using a shorter time window T) reduces ConQuest's rounding error when computing $\lceil \frac{a_i}{T} \rceil$ and $\lfloor \frac{d_i}{T} \rfloor$. Using fewer snapshots (and larger windows) would cause bursts that departed immediately before a_i to be erroneously included in

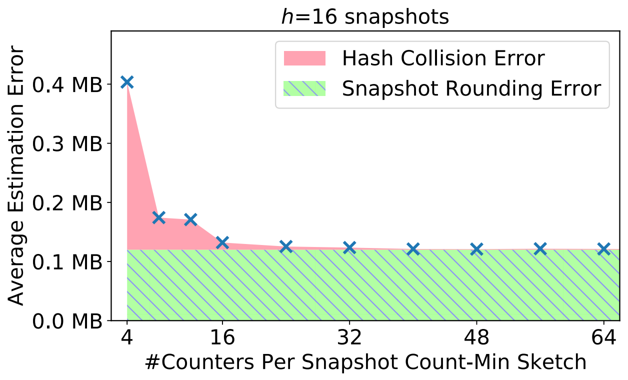


Figure 8: With large CMS, the effect of hash collisions becomes negligible; the flow size estimation error is mainly contributed by snapshot rounding.

the $[a_i, d_i)$ range, thus the rounding error would lead to lower Recall. In the worst case, ConQuest can only look at one snapshot and cannot adapt to the change in queuing delay. As shown in Figure 7, by aggregating h=4 snapshots we can already achieve 93% Recall, and we have diminishing returns after more than h=8 snapshots. Using more snapshots also slightly improves precision. Note that since the maximum queuing delay in the simulated queue is around 6 ms, we configure $T = (6.4/h)$ ms in all combinations, such that aggregating time window from all snapshots can approximately cover the entire queue. These results show that ConQuest can achieve high accuracy once we use enough memory, with diminishing returns for extra resources. The multiple curves in Figure 7 almost overlap, since providing more than enough memory yields negligible difference on Recall, or even slightly decreases Recall; this is because hash collisions lead to over estimations, creating both more false positives and true positives simultaneously.

Flow size estimation error. ConQuest produces an estimate of the size of each flow, not only the largest ones. Such estimations can help network operators analyze the flow size distribution. For example, if there is often only one large flow occupying 90% of the queue during congestion, then it may be sensible to mark or drop the heaviest flow.

As we discussed earlier, the estimated flow size reported by ConQuest is only an approximate, and contains two types of errors: a *snapshot rounding error* is caused by reading an integer number of past snapshots, when in reality the queuing delay may not be integer multiples of snapshot time window size; and a *hash collision error* happens when multiple flow IDs encounter hash collisions, causing the CMS to overestimate flow sizes. In Figure 8, we show ConQuest's average flow size estimation error when using a different number of counters per snapshot. We further separate the effect of hash collision from snapshot rounding by simulating a special version of ConQuest with ideal per-flow counting in each snapshot. As we can see, the error caused by hash collision diminishes quickly with more counters. With C=4 counters per snapshot the effect of hash collision is prominent; with C=64 counters the average flow size estimation error reduces to 120KB, which is mostly caused by snapshot rounding. The interested reader is referred to Appendix B

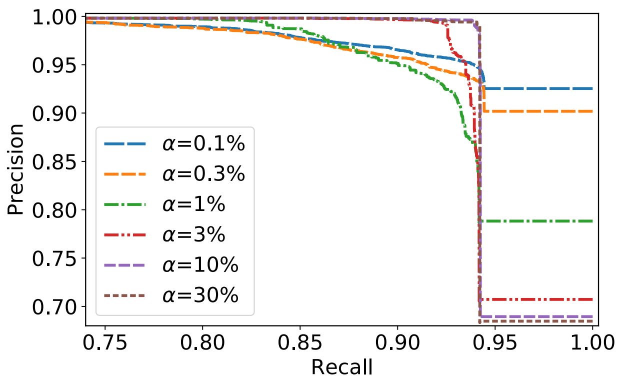


Figure 9: Precision-Recall curve for ConQuest under simulation, for different contributing flow thresholds.

for additional visual analysis of how different parameters affect the estimation error, as described above.

Changing contributing flow threshold. We plot the Precision-Recall curve of ConQuest while changing the contributing flow criteria from α=1% to smaller or larger values, while using a fixed number of h=4 snapshots, each with a R=2, C=8 CMS, creating both collision and rounding error. A smaller α value requires ConQuest to detect heavy flows earlier and report more flows, which is more challenging than reporting only one or two heavy flows when α is very large. As shown in Figure 9, ConQuest can consistently achieve over 90% Precision and Recall while we change α from 0.1% to 30%. In Section 6.3 we perform the same Precision-Recall analysis in a real world prototype and show similar results.

5.2 Closed-Loop Testbed Experiment

We build a testbed experiment to demonstrate ConQuest's potential to analyze and proactively manage queue buildup, by implementing a simple ConQuest-enabled Active Queue Management scheme running at line rate within the data plane. We show that our ConQuest prototype can fit into the hardware constraints of a first-generation PISA programmable switch, and furthermore, we can identify and throttle the flows contributing to congestion to reduce the workload Flow Completion Time. Although the flow-based queue management scheme we implement is primitive and far from optimal, it already demonstrated the potential of future works on building novel AQM schemes using programmable data planes.

Dataset and testbed setup. Our testbed consists of two servers and one Barefoot Tofino Wedge-100 switch. Each server is a 20-core, 100G-memory blade server running Ubuntu 18.04, with Linux kernel version 4.15, and equipped with a Mellanox ConnectX-5 EN 100GbE NIC. We connect both servers to the programmable switch: the *sender* is connected via a 100 Gbps link, while the *receiver* is connected via a 10 Gbps *bottleneck* link. This setup is designed to cause queuing: although one TCP flow can still manage to detect the bottleneck rate correctly, the queue fills up quickly if there are many concurrent TCP flows competing for bandwidth.

We generate workload using the flow-size distribution of a data center Web rack, which are mostly small-to-medium size flows, from the Facebook Data Center Measurement study [30]. The mean and median flow size are 38.8 KB and 1.44 KB, respectively. We schedule flows using exponentially distributed inter-arrival time, choosing λ=155μs (sending one flow every 155μs on average) to achieve 20% average link utilization (2 Gbps) on the 10 Gbps bottleneck link. The sender sends one million workload flows per experiment. The sender also periodically starts one bursty flow per second, and we vary the size of bursty flows between experiments.

All flows are independently managed by the Linux kernel built-in TCP congestion control mechanism, set to New Reno, Cubic, Vegas, DCTCP, or BBR. We bind multiple IP addresses to the receiver and use the SO_REUSEADDR option on the sender to allow sending more than 65,536 concurrent TCP flows. We verified that the servers are not CPU contended.

We tune the baseline, flow-indiscriminative ECN setup to achieve optimal performance, by configuring the switch to mark the ECN bit for outgoing packets when the switch's queue length exceeds the threshold 4096 packets (corresponding to less than 5 ms of queuing delay). We found this to be the minimum queue size needed to allow a single TCP connection to reach line rate, based on the minimum congestion window size required under the Round-Trip Time on our testbed. We also configure the switch to drop packets when the queue length exceeds 16384 packets, although this rarely happens when the sender honors the ECN marking. Our switch is an output-buffered device and we use a single FIFO queue.

Our prototype implementation of ConQuest has h=4 snapshots, each having a CMS with R=2 rows, due to hardware pipeline constraints. We choose C=2048 columns, the largest we can effectively clean within the snapshot clean phase. in total, they use $(R \cdot C \cdot h) \cdot 4$ bytes=65 KB of register memory, less than 1% of the total available on the hardware. The prototype also utilized small fractions of several other hardware resources: it computes R=2 hash functions and performs $(R \cdot h)$=8 memory accesses, both less than 20% of total capacity. This leaves plenty of room for other switch functionality to be run in parallel with ConQuest.

We configure our prototype to rotate the snapshots every T=2 ms, such that aggregating all snapshots will approximately cover the entire queue. We configure congestion threshold to τ=2 ms. When transient congestion is identified, i.e., queuing delay exceeds τ, ConQuest will start marking ECN for flows with $w_f > w_T$=512 packets, corresponding to approximately α=25%.

ConQuest reduces Flow Completion Time. Figure 10 shows the median Flow Completion Time (FCT) of the workload flows, i.e., the small and medium flows generated using the Facebook web rack distribution, with respect to the bytes sent on the bursty flow. Under conventional queue management, when a bursty flow fills up the queue and triggers ECN marking, some packets from small flows inevitably get marked; this is true even if probabilistic (RED-style) ECN marking is used. This leads to a growing FCT for the workload flows as the burst flow becomes larger. Instead, ConQuest only marks packets from the bursty flow, while allowing all small flows to quickly finish sending without being throttled. As a result, the bursty flow has less impact on the FCT of small flows. We have also observed slight improvements in 90%-percentile FCT; however, 99%- and 99.9%-percentile FCT deteriorates since the largest workload flows are also penalized by ConQuest.

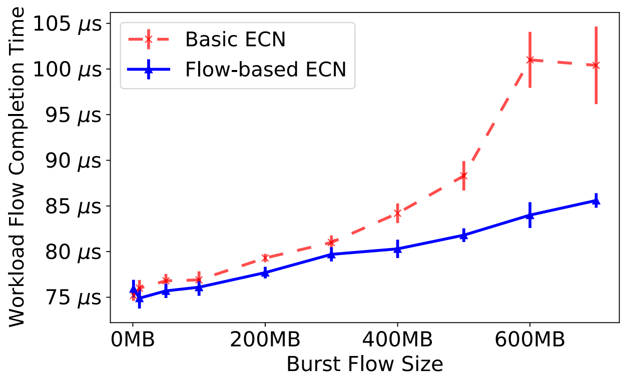


Figure 10: We can configure ConQuest to selectively mark ECN for only the burst flow and not for small flows, leading to better FCT for small flows workload.

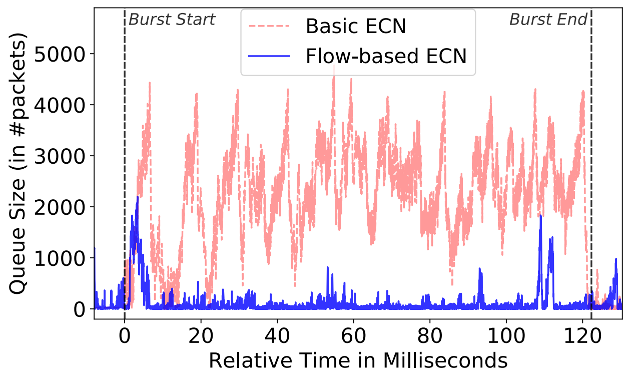


Figure 11: By only marking ECN on contributing flow's packets, ConQuest can effectively throttle the bursty flow and maintain a shorter queue length.

The result shown in Figure 10 was performed using New Reno congestion control (ConQuest reduced median FCT by 6.9%); experiment results for other congestion control algorithms are similar (ConQuest reduced median FCT by: 7.3% for Cubic, 1.6% for Vegas, 0.56% for DCTCP), except for BBR. Notably, BBR does not honor ECN marking (or dropping) as the congestion control signal and performs its own queue buffer utilization probing, and therefore ConQuest cannot affect its sending rate.

Figure 11 shows the queue length statistics we collected from the programmable switch, while a 50 MB burst flow interacts with the small flow workload. Under regular ECN settings the queue is quickly filled up to the ECN marking threshold, at which point all flows are subject to congestion control and queue length oscillates, until the bursty flow finishes sending. In contrast, when we enable flow-based ECN by querying ConQuest, the bursty flow is quickly throttled and the queue remains short during the entire sending period. Note that the queue length has many short spikes when ConQuest is enabled; this is because multiple short flows can all quickly finish without being marked or dropped.

Our results show that it is possible to improve network performance at the switch level with flow-level queuing analysis and queue buildup mitigation. Although the AQM scheme we implement with ConQuest in the testbed is very primitive, it already demonstrates the potential performance improvements of using programmable switches to implement sophisticated AQM algorithms. We note that in practical networks such as wide-area / carrier networks, merely adding an ECN flag cannot throttle flows immediately and effectively; we need to take other actions on the packets of contributing flows, such as dropping, rerouting, or scheduling them in a separate queue.

6 CONQUEST FOR LEGACY DEVICES

Legacy (i.e., non-programmable) routers are not designed for precise queuing analysis. They often only support polling the total queue length statistics at a coarse time interval, providing no insight into which flows occupy the queue. Existing networks are not going to replace legacy routers with programmable switches overnight. Yet advanced fine-grained queue monitoring techniques are necessary today, both for debugging existing devices and for understanding the buffer capacity needed to support their operational workload. This is especially true in carrier networks, where the upgrade cycles for network equipment are longer and network operators cannot perform measurements at end hosts. Therefore, network operators have been looking for ways to use an advanced programmable switch as a plug-in debugging tool, to measure and analyze queuing on legacy routers in their network wherever problems arise.

We propose a novel way to use ConQuest as a tool for *selectively* monitoring one legacy router, *temporarily*, in a non-intrusive manner, by tapping its existing ingress and egress links and using a programmable switch to process the tapped traffic. With one programmable switch at hand, network operators can debug any legacy device in the network, gaining on-demand visibility into its queuing dynamics and congestion in real time, without having to replace the device with a programmable one. Tapping is often readily available at the physical layer (split-fiber), or as a monitoring capability provided by the equipment vendor.

We deployed our extended prototype of ConQuest in two different settings: tapping into a border router in a campus network[1], and tapping into a carrier-grade router in an ISP testbed.

At Princeton University, our campus network operator had identified one border router that occasionally suffers from massive packet drops under low average link utilization. We suspect transient congestion is taking place, however existing diagnostic tools only report queue buffer utilization (alongside other metrics) at minute-level granularity, without apparent anomaly. We helped our campus network operator to use a programmable switch running ConQuest to tap and analyze this border router's ingress and egress traffic, and successfully located the cause of the drops: a performance monitoring tool that failed to schedule throughput tests in series (as claimed), creating incast from multiple senders across Internet2. The queuing delay oscillates wildly from empty to full, and there are 4-5 contributing flows in the queue, all for

[1] The diagnostic process involves no access to personal data and has been approved by our university's Institutional Review Board.

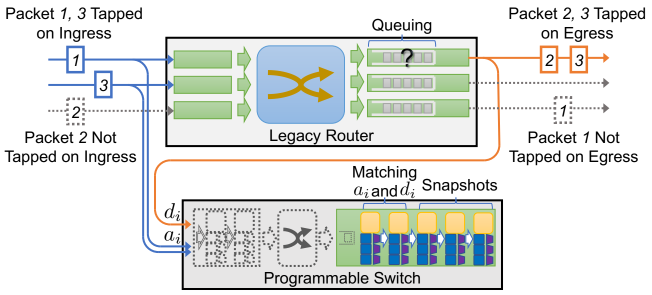

Packet 1, 3 Tapped on Ingress
Queuing
Packet 2, 3 Tapped on Egress
Packet 2 Not Tapped on Ingress
Legacy Router
Packet 1 Not Tapped on Egress
Matching a_i and d_i
Snapshots
d_i
a_i
Programmable Switch

Figure 12: Using a PISA switch to analyze queuing in a legacy router, by tapping ingress and egress links.

active throughput testing. Here we see that passive monitoring powered by ConQuest was able to diagnose the performance problems caused (ironically!) by an active performance monitoring tool.

At AT&T Labs, we use a Cisco carrier-grade router to process synthetic bursty traffic, and let ConQuest analyze tapped traffic to verify its accuracy and robustness under the tapping setup. We present the details of our testbed and the results in Section 6.3.

6.1 Tap Multiple Links of Legacy Router

Figure 12 illustrates the setup for using a PISA switch to monitor queuing in a legacy router. We tap a subset of the legacy router's ingress and egress ports and mirror their traffic to ports with a common packet-processing pipeline in the PISA switch. Ideally, we would like to tap all ingress and egress links; however, this may be impractical due to cost or tapping link availability; nevertheless we can analyze the legacy router's queuing efficiently even by tapping only a subset of the links, as we discussed in Section 6.2.

The PISA switch records the arrival timestamp (a_i) of a packet when it appears in a tapped ingress link, and records the departure timestamp (d_i) of a packet when it appears in a tapped egress link. To recover accurate and unbiased queuing delay ($d_i - a_i$), the tapping links for the ingress and egress ports should have equal and constant latency.

6.2 Match Ingress and Egress Packets

To recover the queuing delay ($d_i - a_i$) experienced by packet i, we would like to match the appearances of packet i in both the tapped ingress link and the tapped egress link. There are several technical details to consider:

Hash digest. We hash a packet's header fields to obtain a hash digest for efficiently matching a packet's appearance on a tapped egress link with its earlier appearance on the ingress link. For IPv4 packets, we can examine the IPID field. For TCP packets, we can also observe the sequence/acknowledgement number to distinguish individual packets within the same flow. Matching IPv6/UDP packets is more challenging and we omit the implementation details.

Storage and timeout. The digest and arrival time a_i from the tapped ingress are first inserted to a hash-indexed array. Later, when a copy of the same packet appears on the egress tapping link at time d_i, we compute the same digest to fetch a_i from the array and compute the queuing delay ($d_i - a_i$), and also clear the entry from the array.

Not seen on egress: Some packets that appear on a tapped ingress link may be dropped or routed to an untapped egress port; therefore, they never appear on the tapped egress link. For example, in Figure 12, packet *1* was tapped on an ingress link, but was routed to an egress port not being tapped. These packets would fill up the register array that would never be matched and are therefore useless. We solve this issue by implicitly expiring entries: we allow an entry to be evicted from the array once its arrival timestamp has aged more than the maximum possible queuing delay, and can thus be considered expired.

Not seen on ingress. A packet that arrives at the tapped egress link may not have a corresponding digest and arrival timestamp stored in memory. This may occur if the packet entered the router from an untapped ingress link, or a failed insertion to the array due to hash collision. For example, in Figure 12 packet *2* comes in from an untapped ingress port, but appears on the tapped egress port, so d_2 is known but a_2 is unknown. We cannot query if these packets belong to a contributing flow; however, we still insert them into the current snapshot using the departure timestamp, since they contributed to the congestion at our monitored egress port.

6.3 Validation with a Cisco CRS Router

We built a tapping testbed to evaluate if ConQuest can accurately diagnose queuing in a legacy switch. We use a programmable Barefoot Tofino Wedge-100 switch ("programmable switch") to tap 3 ingress links and 1 egress link of a Cisco CRS 16-Slot Single-Shelf System ("legacy router"), all running at 10 Gbps. We use an IXIA traffic generator to feed traffic into the 3 ingress ports; the legacy switch is configured to route all traffic to the same egress port, into a single FIFO queue.

We extend the ConQuest P4 program to match ingress and egress packets to calculate queuing delay, and compute ground truth statistics for evaluation purpose. The combined P4 program has around $1,200$ lines of code. We have verified that our extended P4 prototype is indeed accurately measuring the queuing delay in the legacy router (see Appendix A).

We configure the IXIA traffic generator to send 10 flows as background workload, ranging from 1 Mbps to 5 Gbps, and send 3 periodically bursty flows, with varying burst duration from 50 µs to 5 ms. Note that the number of flows are limited by our need to maintain ground truth per-flow counters for evaluation purpose, and ConQuest itself can work with a large number of flows, as demonstrated in Section 5. Since the prototype has h=4 snapshots and the maximum observed queuing delay is around 4 ms, we configure the ConQuest to use snapshot interval T=1 ms. Each snapshot uses a R=2 row, C=64 column Count-Min Sketch, which is large enough to not cause any hash collision. The congestion reporting threshold is set to τ=0.5 ms, about 1/8 of maximum queue length, similar to previous experiments.

We compared the reported packets ConQuest identified as part of a contributing flow to the per-packet ground truths we fetched from IXIA and our extended P4 prototype, and computed Precision and Recall metrics. Figure 13 shows the Precision-Recall curve, under different contributing flow criteria α. ConQuest consistently achieves over 90% Precision and Recall when identifying contributing flows, for α ranging from 0.1% to 30%. Although we cannot

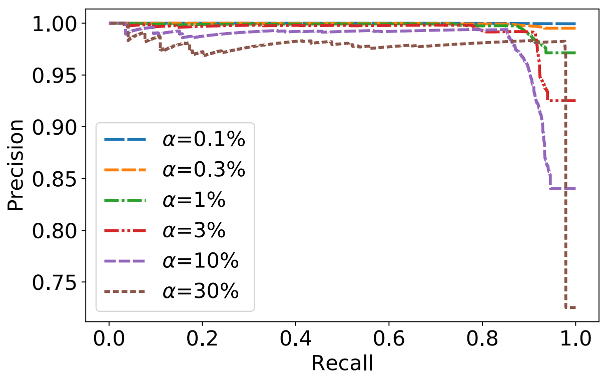


Figure 13: Precision-Recall curve for ConQuest's P4 prototype under tapping setup.

support taking immediate corrective action, ConQuest still provides us unprecedented visibility and high accuracy for analyzing queuing in a tapped legacy router.

7 RELATED WORK

Measuring queue buildups. Zhang et al. [38] implemented a high-precision microburst measurement framework in data-center networks, by polling multiple switches' queue depth counter at high frequency, and analyzing duration and inter-arrival time of microbursts. However, the system provides limited insight into the contents of the queue, such as which flows contributed to a microburst or the flow size distribution during the queue buildup. Several recent papers use programmable switches for fine-grain logging of traffic in the data plane. SpeedLight [36] is a general system for recording synchronized traffic statistics across multiple switches for offline analysis, including analyzing queuing dynamics. BurstRadar [20] can log packets in a ring buffer at a single switch during queue buildup for offline analysis. *Flow [32] compresses the packet logs before exporting the measurement data to reduce overhead on the remote hosts. Speedlight, BurstRadar and *Flow all provide fine-grained measurement data for *offline* analysis, but cannot identify or act on contributing flows directly in data plane. Meanwhile, HPCC [24] measured switch queue length to improve end-host based congestion control.

Data-center traffic management. In recent years there has been much work on alleviating congestion in data centers. For example, load-balancing schemes like Presto [19], DRILL [17] and CONGA [2] disperse the offered load over multiple paths, without addressing the *root cause* of queue buildup. In contrast, ConQuest enables the switches to identify and target individual flows contributing to backlogged queues. Meanwhile, data-center transport protocols such as NDP [18] and Homa [26] reduce queuing delay at switches, but they typically assume the end-host network stack (e.g., tenant VM) participates *honestly* in the protocol, or require enforcement by the underlying hypervisor or NIC. Fastpass [29] offers a centralized traffic orchestration approach for preventing queue buildup, by centrally allocating the capacity of network links to individual senders. Meanwhile, ConQuest does not impose any additional mechanisms or overheads on the end-host network stack, hypervisor, or NIC. This is especially critical for transit and enterprise networks that do not have control over the end hosts.

Fair queuing. Sharma et al. [31] proposed an approximate per-flow fair queuing mechanism using programmable switches, which reduces the bursty flow's impact on other traffic. Instead of enforcing fairness among all flows, ConQuest identifies individual flows contributing to queue backlogs, and therefore enables acting directly on those flows.

Estimating FIFO queue state. Queue Inference Engine, by Larson [22] and later improved in [8], is an algorithm to analyze the queuing state of a FIFO queue, with random arrivals following a Poisson process. QIE only uses departure timestamps as input, and can infer queuing delay based on observing consecutive departures ("busy periods"). Instead, ConQuest calculates the exact queuing delay for each packet using queuing metadata, and its goal is to analyze the heavy flows in the queue. ConQuest can be used under arbitrary packet arrival time distributions, such as microbursts.

Sliding window query. Our data structure contributes to a body of theoretical work on streaming algorithms on sliding time windows. For example, several works [1, 6, 10, 37] propose algorithms for set membership or heavy-hitter queries on a *fixed-size* sliding window. In contrast, our work deals with a *dynamic* query window $[a_i, d_i)$, which varies across the packets in the stream. As such, the window sizes of future queries are unknown when a packet enters the data structure. ConQuest addresses this challenge by reading from a variable number of time-window snapshots. Ben Basat et al. has explored a similar dynamic window query problem in [5] and proposed advanced data structures that run on general purpose computers. To the best of our knowledge, ConQuest is the first solution to be implemented within the resource constraints of programmable switch hardware.

8 CONCLUSION

We present ConQuest, a scalable data structure for analyzing queuing in network switches in real time. ConQuest reports which flows contribute to the queue buildup, and enables direct per-packet action in the data plane. We implement a ConQuest prototype on a programmable hardware switch using only 65 KB of register memory. Testbed evaluation demonstrates ConQuest can effectively identify the contributing flows, and enable the switch to throttle them. In addition, we propose a novel way to use ConQuest to monitor queuing in legacy network switches. In our ongoing work, we are deploying ConQuest in both a carrier and a campus network to diagnose performance problems in legacy devices—as a first step in demonstrating the benefits of data-plane queue measurement to network operators.

Acknowledgements: This research is supported by NSF grant CCF-1535948, a gift from AT&T, and The Eric and Wendy Schmidt Fund for Strategic Innovation. We would like to thank Dr. Hyojoon Kim from Princeton's Office of Information Technology for deploying our prototype in Princeton University's campus network. We sincerely thank the anonymous reviewers of CoNEXT'19 and our shepherd Prof. Marco Mellia for their helpful comments and feedback.

REFERENCES

[1] Yehuda Afek, Anat Bremler-Barr, Shir Landau Feibish, and Liron Schiff. 2018. Detecting Heavy Flows in the SDN Match and Action Model. *Computer Networks* 136 (2018), 1–12.

[2] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. 2014. CONGA: Distributed congestion-aware load balancing for datacenters. In *ACM SIGCOMM Conference.* 503–514.

[3] Mohammad Alizadeh, Albert G. Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center TCP (DCTCP). In *ACM SIGCOMM Conference.* 63–74.

[4] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. 2004. Sizing router buffers. In *ACM SIGCOMM Conference.* 281–292.

[5] Ran Ben Basat, Roy Friedman, and Rana Shahout. 2018. Stream frequency over interval queries. *Proceedings of the VLDB Endowment* 12, 4 (2018), 433–445.

[6] Ran Ben-Basat, Gil Einziger, Isaac Keslassy, Ariel Orda, Shay Vargaftik, and Erez Waisbard. 2018. Memento: Making sliding windows efficient for heavy hitters. In *ACM CoNEXT Conference.* 254–266.

[7] Theophilus Benson, Aditya Akella, and David A. Maltz. 2010. Network traffic characteristics of data centers in the wild. In *ACM SIGCOMM Internet Measurement Conference.* 267–280.

[8] Dimitris J Bertsimas and Leslie David Servi. 1992. Deducing queueing from transactional data: the queue inference engine, revisited. *Operations Research* 40, 3-supplement-2 (1992), S217–S228.

[9] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM Conference.* 99–110.

[10] Vladimir Braverman, Ran Gelles, and Rafail Ostrovsky. 2014. How to catch L_2-heavy-hitters on sliding windows. *Theoretical Computer Science* 554 (2014), 82–94.

[11] Yanpei Chen, Rean Griffiths, David Zats, Anthony D. Joseph, and Randy H. Katz. 2012. Understanding TCP Incast and its Implications for Big Data Workloads. *;login* 37, 3 (June 2012).

[12] Mosharaf Chowdhury and Ion Stoica. 2012. Coflow: a networking abstraction for cluster applications.. In *HotNets*. 31–36.

[13] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient coflow scheduling with varys. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 443–454.

[14] Mikkel Christiansen, Kevin Jeffay, David Ott, and F. Donelson Smith. 2001. Tuning RED for Web traffic. *IEEE/ACM Transactions on Networking* 9, 3 (2001), 249–264.

[15] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: The Count-Min Sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[16] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking (ToN)* 1, 4 (1993), 397–413.

[17] Soudeh Ghorbani, Zibin Yang, Philip Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. 2017. DRILL: Micro Load Balancing for Low-latency Data Center Networks. In *ACM SIGCOMM Conference.* 225–238.

[18] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *ACM SIGCOMM Conference.* 29–42.

[19] Keqiang He, Eric Rozner, Kanak Agarwal, Wes Felter, John Carter, and Aditya Akella. 2015. Presto: Edge-based load balancing for fast datacenter networks. In *ACM SIGCOMM Conference.* 465–478.

[20] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. 2018. BurstRadar: Practical Real-time Microburst Monitoring for Datacenter Networks. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys)*.

[21] Aleksandar Kuzmanovic and Edward W. Knightly. 2003. Low-rate TCP-targeted denial of service attacks: The shrew vs. the mice and elephants. In *ACM SIGCOMM Conference.* 75–86.

[22] Richard C Larson. 1990. The queue inference engine: Deducing queue statistics from transactional data. *Management Science* 36, 5 (1990), 586–601.

[23] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: a better NetFlow for data centers. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 311–324.

[24] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: high precision congestion control. In *ACM SIGCOMM Conference.* ACM, 44–58.

[25] Yi Lu, Andrea Montanari, Balaji Prabhakar, Sarang Dharmapurikar, and Abdul Kabbani. 2008. Counter braids: A novel counter architecture for per-flow measurement. *ACM SIGMETRICS Performance Evaluation Review* 36, 1 (2008), 121–132.

[26] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *ACM SIGCOMM Conference.* 221–235.

[27] Wladyslaw Olesinski and Steve Driediger. 2009. Fair WRED for TCP UDP traffic mix. (2009). US Patent 7,616,573.

[28] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis. 2000. CHOKe-A stateless active queue management scheme for approximating fair bandwidth allocation. In *IEEE INFOCOM.* 942–951.

[29] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2014. Fastpass: A centralized "zero-queue" datacenter network. In *ACM SIGCOMM Conference.* 307–318.

[30] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *ACM SIGCOMM Conference*, Vol. 45. 123–137.

[31] Naveen Kr. Sharma, Ming Liu, Kishore Atreya, and Arvind Krishnamurthy. 2018. Approximating Fair Queueing on Reconfigurable Switches. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 1–16.

[32] John Sonchack, Oliver Michel, Adam J. Aviv, Eric Keller, and Jonathan M. Smith. 2018. Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With *Flow. In *USENIX Annual Technical Conference.* 823–835.

[33] The P4 Language Consortium. 2018. P4$_{16}$ Language Specification. https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.html. (Nov. 2018).

[34] Vladimir Gurevich for Barefoot Networks. 2017. Programmable Data Plane at Terabit Speeds. https://p4.org/assets/p4_d2_2017_programmable_data_plane_at_terabit_speeds.pdf. (May 2017).

[35] Damon Wischik and Nick McKeown. 2005. Part I: Buffer Sizes for Core Routers. *ACM SIGCOMM Computer Communication Review* 35, 3 (2005), 75–78.

[36] Nofel Yaseen, John Sonchack, and Vincent Liu. 2018. Synchronized network snapshots. In *ACM SIGCOMM Conference.* 402–416.

[37] MyungKeun Yoon. 2010. Aging bloom filter with two active buffers for dynamic sets. *IEEE Transactions on Knowledge and Data Engineering* 22, 1 (2010), 134–138.

[38] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. 2017. High-resolution measurement of data center microbursts. In *ACM SIGCOMM Internet Measurement Conference.* 78–85.

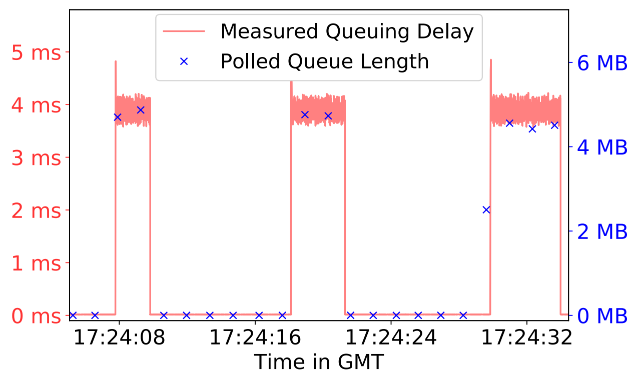


Figure 14: Queuing delay measured by our prototype matches the ground-truth queue-depth reported by the legacy switch.

APPENDIX

A MEASURING DELAY WITH TAPPING

We validated that our extended ConQuest prototype correctly estimates the queuing delay in a legacy switch, by comparing the queuing delay estimated by ConQuest with the ground truth queue length reported by the legacy switch. We send periodically bursty traffic into the legacy switch to create queuing. As shown in Figure 14, the queuing delay computed by our P4 program nicely aligns with the queue length reported by the legacy switch (divided by line rate 10 Gbps).

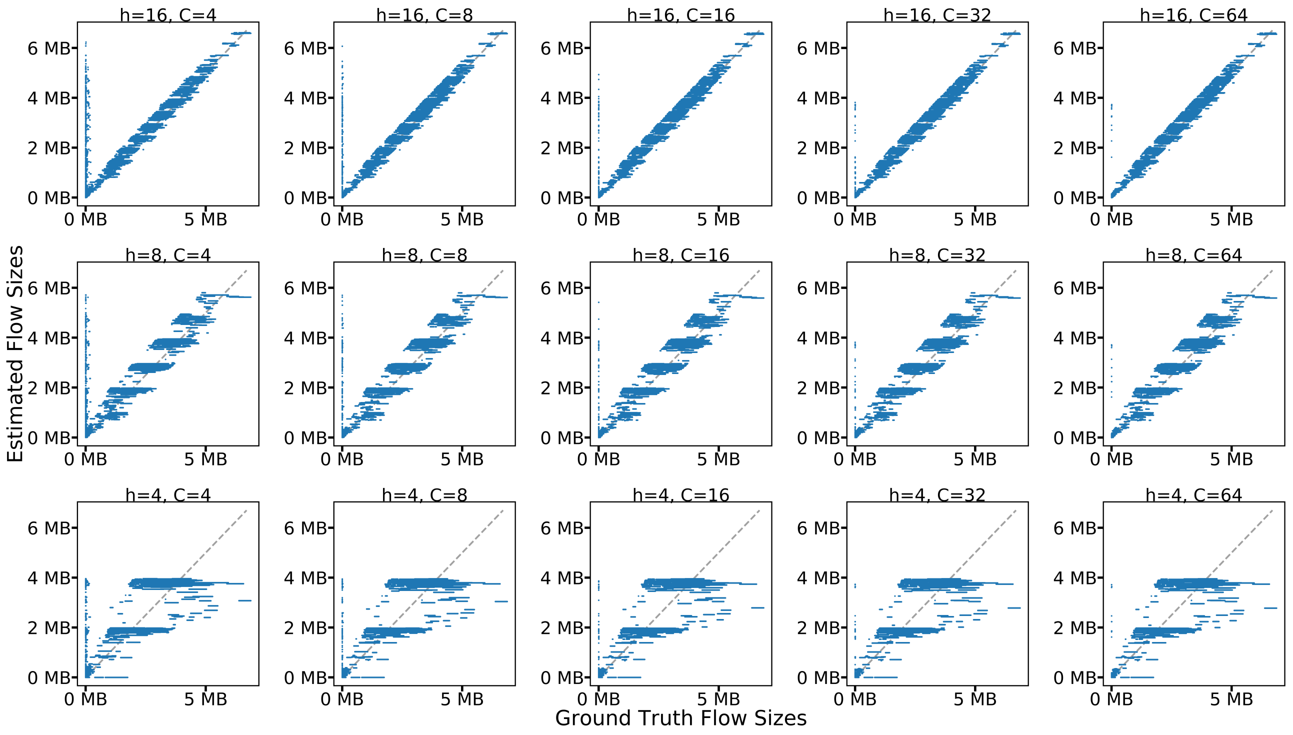
B ESTIMATION ERROR ANALYSIS

As we discussed earlier in Section 5.1, ConQuest incurs two kinds of error when estimating the size of a flow in the queue: rounding error due to querying integer number of snapshots, and overestimation by Count-Min Sketch due to hash collisions with other flow IDs.

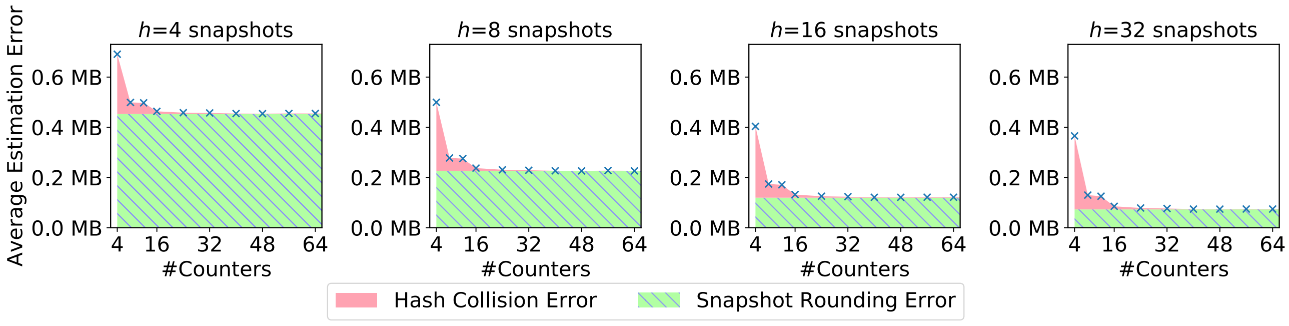
To further illustrate the two kinds of errors, we draw scatter plots of ground truth flow sizes versus estimated flow sizes reported by ConQuest in our simulation experiments. In Figure 15a, we first notice that the plots on the lower rows use fewer snapshots, hence a ladder-shaped rounding effect (due to querying integer number of recent snapshots) is prominent, while using more snapshots the estimation can have higher accuracy (closer to $y = x$ line). Meanwhile, the plots to the left use smaller CMS, causing some small flows (with ground truth flow size close to zero) to collide with larger flows, and the overestimation caused by such hash collisions is shown as dots close to the y-axis. When using larger CMS, these hash collisions start to diminish.

We can quantify the effects of the two kinds of errors by computing the average flow size estimation error under various configurations. In Figure 15b, we plot the average value of absolute flow-size estimation error under different configurations. We can see that the average estimation error is 120 KB for h=16 snapshots, with R=2, C=16 CMS. When using only h=4 snapshots and R=2, C=16, the average and median estimation error grows to 461 KB and 281 KB respectively. As a reference, under h=4 snapshots (T=1.6 ms), 10 Gbps line rate setup, the total traffic recorded in each time window is 2 MB.

We separate the effect of rounding versus hash collision by simulating a special version of ConQuest that does not use CMS and records exact flow sizes in snapshots, and attribute its error to rounding (plotted in shaded green). As we can see from Figure 15b, the error caused by hash collisions diminished quickly with more counters in CMS; when ConQuest is running with adequate memory, the estimation errors are mainly caused by snapshot rounding error, and we shall note that such error will not cause significant impact for accurately identifying heavy flows.



(a) Scatter plot of estimated flow size vs. ground truth, using different number of snapshots h and different CMS width C.



(b) Attributing estimation errors to flow ID hash collisions and snapshot rounding errors.

Figure 15: Analyzing flow size estimation errors.