F3: Fast and Flexible Network Telemetry with an FPGA coprocessor

WEIQI FENG, Harvard University, USA
JIAQI GAO, Alibaba, USA
XIAOQI CHEN, Purdue University, USA
GIANNI ANTICHI, Politecnico di Milano, Italy
RAN BEN BASAT, University College London, UK
MICHAEL MINGCHAO SHAO, Meta, USA
YING ZHANG, Meta, USA
MINLAN YU, Harvard University, USA

Traffic monitoring in the dataplane is vital for reacting to network events such as microbursts, incast, and attacks. However, current solutions are constrained by the limited resources available on modern ASICs and don't provide the flexibility required to identify repeating patterns, such as applications whose flows communicate with a server at regular intervals. While such flexibility can be achieved using a co-processing CPU, it is generally too slow to provide insights quickly enough. In this paper, we show how an FPGA co-processor placed alongside the switching pipeline enables flexible traffic monitoring at data plane rates. While FPGAs have large memory and expressive processing, their throughput is significantly lower than switch ASICs. To bridge the throughput gap, we split query execution between the switch and FPGA and present methods that prevent processing all packets in FPGA. As a result, our system misses up to 5.0x fewer DDoS attack vectors than ACC-Turbo, the state-of-the-art on-switch solution, and up to 24% fewer microburst-contributing flows for the same precision rate.

CCS Concepts: • **Networks → Network measurement**.

Additional Key Words and Phrases: Network Monitoring, Programmable Network

ACM Reference Format:
Weiqi Feng, Jiaqi Gao, Xiaoqi Chen, Gianni Antichi, Ran Ben Basat, Michael Mingchao Shao, Ying Zhang, and Minlan Yu. 2024. F3: Fast and Flexible Network Telemetry with an FPGA coprocessor. *Proc. ACM Netw.* 2, CoNEXT4, Article 25 (December 2024), 22 pages. https://doi.org/10.1145/3696397

1 Introduction

Monitoring systems are key components in datacenter networks [11, 13, 19, 26, 43, 51, 85, 86, 89]. As networks grow in size, it is increasingly important to deploy complex analysis solutions and correlate data across times and flows. Such analysis helps us to gain insights into recurrent events such as microbursts (which affect application performance) [19, 38, 76]; advanced visibility is also the key enabler to support fine-grained automated network control [13, 33, 51] and detailed troubleshooting [28, 35, 73].

Authors' Contact Information: Weiqi Feng, Harvard University, USA; Jiaqi Gao, Alibaba, USA; Xiaoqi Chen, Purdue University, USA; Gianni Antichi, Politecnico di Milano, Italy; Ran Ben Basat, University College London, UK; Michael Mingchao Shao, Meta, USA; Ying Zhang, Meta, USA; Minlan Yu, Harvard University, USA.

ACM 2834-5509/2024/12-ART25
https://doi.org/10.1145/3696397

Some existing works directly analyze traffic at programmable switches [19, 44]. However, state-of-the-art programmable switches have limited capabilities and memory that have to be used for essential functions such as ACL rules, customized forwarding [69], load balancing [13, 26, 39], congestion control [51], and other network functions and applications [31, 36, 55]. As a consequence, state-of-the-art solutions that leverage in-network computing for monitoring resort to enabling only a few simple queries at run-time, effectively limiting network visibility [29, 59]. For example, we can identify the repeated incast by the same application by detecting repetitive transmission patterns, e.g., due to an aggregation step by a distributed training process [61], a functionality that is hard to achieve on the switch alone due to its complexity. While one can use the switch's CPU to detect such a pattern [80], it has a low bandwidth and is needed for various other functionalities such as installing rules [44].

Other solutions advocate for analyzing data reported by switches at collectors [11, 29, 52, 90], dedicated servers located in ordinary racks within the datacenter fabric [34, 56]. This is because they provide plenty of cheap memory, and their general-purpose CPUs are well equipped to run data analysis tasks [29]. However, using a remote CPU markedly slows down the control loop, which is prohibitive for detecting attacks in real time. Further, every single switch can generate up to millions of telemetry data reports per second [59, 93] and a datacenter network comprises hundreds of thousands of them [28]. As the amount of data keeps growing with larger networks and higher line rates [67], it is becoming increasingly hard to scale data collection and analysis in this way [40, 75, 93].

A clear trade-off emerges: collectors provide the flexibility to run complex data analysis but not at data-plane speed, which is essential to enable a fast reaction to ephemeral events [19]. On the other hand, state-of-the-art programmable switches provide high processing speed and enable quick control loops in the data plane, but lack resources and flexibility.

In this paper, we propose F3, a fast and flexible FPGA-assisted analysis telemetry system. We explore using FPGAs that are located alongside the switching ASIC pipeline as a co-processor to support comprehensive analysis functions for network telemetry at data-plane speed. FPGAs are becoming a commodity in the datacenter infrastructure [21, 23, 47, 48, 83, 88] and vendors start to package FPGA as a co-processor attached to the switch ASIC [3, 8]. Using this resource available for network telemetry brings several advantages: (1) FPGAs guarantee high throughput and low latency analysis: modern chips can handle hundreds of Gbps input data while achieving deterministic μs level processing latency; (2) Modern FPGAs are more flexible than commodity programmable ASICs: they have orders of magnitude more memory (e.g., High Bandwidth Memory is multiple GBs), thousands of arithmetic units and they can be as generic as a high-end CPUs [79]. This makes them the perfect fit for performing advanced monitoring analysis; (3) FPGAs, located alongside the switch pipeline, allow for fast reaction to events while freeing expensive switch resources that can be used for packet-processing operations; (4) It is possible to reprogram slices of FPGA and change its traffic analysis behavior without downtime.

The challenge for F3 is bridging the throughput gap between the sheer amount of data processed by switches (terabit scale) and the more limited capabilities of FPGAs (hundreds of Gbps), with no to minimal downtime due to switch recompilations. To overcome this main challenge, we first propose to decompose monitoring queries, where the switch performs the necessary *throughput reduction functions* and FPGA performs the core analysis. Second, we propose to use *runtime shaping* that can help in controlling the amount of data sent from the switch to FPGA. Finally, we developed several FPGA-based traffic analysis modules (i.e., change detection, hierarchical counter aggregation) that can be used as a library by operators and that can be extended if needed.

We implemented F3 using a combination of a commodity programmable switch pipeline and a Xilinx Alveo U250 card placed alongside it to emulate a single device. Using real network traces

that include 11 different attack types, such as SYN Flood and Portscan, we show that F3 misses up to 5.0x fewer DDoS attack vectors than ACC-Turbo. Additionally, using real flow-size distributions from datacenter network traffic, we show that our solution misses up to 24% fewer microburst-contributing. Finally, we show that F3 requires minimal switch and FPGA overheads. Namely, it requires at most 2.1% of the switch's SRAM and 26.9% FPGA's BRAM for all the proposed queries while being able to detect all lossy flows with a 99.99% probability with just an additional 16.5% of FPGA's BRAM.
This work does not raise any ethical issues.

2 Motivation

In this section, we discuss the need for smarter analysis and better performance for network monitoring systems and why FPGA is a great fit.

2.1 The need for comprehensive analysis

Table 1 lists some example network measurement queries that require comprehensive analysis. Some require more resources than are available on modern switches, while others are implementable on switches but have accuracy that can be improved given more memory. We use three examples to showcase important dimensions of complexity: analyzing across time, across flows, and computing composite functions.

Example	Temporal/Contextual	Analysis Functions
Microburst analysis [19, 38, 76]	Find a group of 5-tuples that increase throughput at sub-millisecond granularity	Change pattern detection
LDoS attack detection [45, 92]	Find 5-tuples that periodically send traffic with a period of RTO level	AutoCorrelation
Priority flow contention [72, 73, 76]	Find 5-tuples with low priority that consume lots of bandwidth	Thresholding
Metastable failure [7]	Find 5-tuples that last a long time and keep having packet loss	Causal inference
Pulse-wave DDoS Analysis [12]	Find a group of 5-tuples perform puls-wave DDoS attack	Clustering algorithm
Super spreaders [81]	Find source IPs that send 5-tuples to more than threshold number of destination IPs	Thresholding
Port scan [37]	Find source IPs that send 5-tuple to more than threshold number of destination ports	Thresholding
Newly opened TCP Conns [82]	Find source IPs that send more than threshold number of SYN packets	Thresholding
Slowloris attack detection [30]	Find a group of 5-tuples from the same source that sends low volume traffic	Thresholding
Packet loss detection [50]	Find 5-tuples that lost packets and the number of lost packets	Flow-level loss detection

Table 1. Analysis table for common network problems.

Temporal analysis. One good example that is common in wide-area networks is the detection of low-rate denial of service (LDoS) attacks [45, 92]. In a specific type of LDoS attack [45], the attacker sends short bursts lasting around the duration of a round-trip time (RTT), with intervals between bursts on the scale of a retransmission timeout (RTO). Even a single DoS stream can provoke the victim flow to repeatedly enter the retransmission state, and such attack traffic is challenging to detect, given their small volume.

To catch the suspects, we need to look into each flow's past traffic volume and identify the interval patterns, using techniques like auto-correlation as in previous work [20]. As shown in [45], different LDoS attacker flows can have different lengths of burst periods. Since we do not know the attacker flow's burst period beforehand, we can calculate an auto-correlation score for possible lengths. To exemplify the problem, we use a simple topology in *ns-3* with three senders, one receiver, and one attacker connected by a single switch through 10G links. To simulate datacenter traffic traversing different paths in a 3-tier topology, we assign senders RTTs of $100\mu s$, $200\mu s$, and $300\mu s$. We use commonly used RTO value of $1ms$ for datacenter traffic [63]. The aggregate background TCP traffic throughput from all senders is 10Gbps and the flow size is sampled from DCTCP flow size CDF [14]. The attacker creates one low-rate square-wave DoS flow by sending 1000 64-byte UDP packets every RTO. The resulting average DoS flow throughput is only 0.512 Gbps. We record the #bytes received by the switch for each flow every epoch with different epoch granularity ($100\mu s$, $200\mu s$, $400\mu s$).

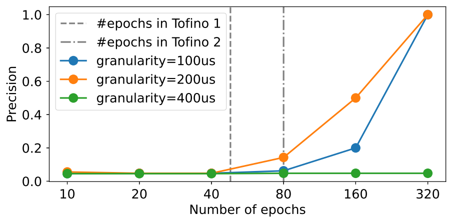

Fig. 1. LDoS Attack Detection Precision

Figure 1 shows the LDoS attack detection precision by running an auto-correlation algorithm with a lagged time interval of retransmission timeout (RTO) under different numbers of historical epochs. The LDoS attack detection accuracy increases with more history epochs and reaches to 100% with 320 epochs. However, a programmable switch can't maintain data about hundreds of historical epochs due to its *stage constraints*. Even the most advanced programmable switch [4] has only 20 pipeline stages and allows at most 4 register arrays per stage, translating to access to a maximum of 80 epoch data in the pipeline. Also, performing auto-correlation on historical epoch data costs extra pipeline stages on a programmable switch. Although programmable can recirculate packets through the pipeline multiple times, it reduces the pipeline bandwidth available to process packets [70].

Compared with programmable switches, FPGA has no pipeline stage constraints. Algorithms with dependency (e.g., auto-correlation) can be synthesized and implemented into a customized circuit on the FPGA. Then the number of epoch data it can access is only constrained by FPGA's memory capacity. Modern FPGA has lots of on-chip memory resources (e.g., high-bandwidth memory is multiple GBs [9]), and it can keep enough history information for analysis.

Contextual analysis. In the datacenter network, there are many applications, each consisting of multiple flows, running concurrently and competing for network bandwidth. Some applications (e.g., machine learning training) are very bursty and might thus negatively affect the behaviors of others. This is because microbursts and incast are known to degrade datacenter performance [14, 32]. It is thus important to analyze flows based on context (e.g., belonging to which application or coming from which subnet) to troubleshoot the interactions between application behavior and the network.

One such use case is finding the root cause of microbursts. Often caused by incast, a microburst is a phenomenon where a switch's queue is suddenly overwhelmed by burst traffic within a short timescale. Previously, ConQuest[19] analyzed each flow's weight in the queuing buffer and identified flows with weights larger than a threshold as the contributing flows of the microburst. However, such a crude analysis fails to recognize when a microburst is caused by a group of many flows sharing the same destination, since each flow may not exceed the threshold. As shown in [62], flows causing microbursts can come from several particular subnets. And we need to aggregate flows based on the subnet context to observe the bursty throughput increase. To illustrate this issue, we use a spine-leaf topology in *ns3*, with the detailed setup described in §5.1.2. Figure 2 shows the per-flow throughput and aggregated per-subnet (IP/24) throughput during microbursts. Like the incast setting in [51], the microburst is created by randomly selecting 60 senders and one receiver, each sending 500KB. From the per-flow throughput view, it's hard to distinguish the incast flows from the background ones. However, when looking from the IP-subnet level, it's clear that TCP incast flows originate from 4 different subnets, which means 4 different racks in the simulated network topology. It's worth mentioning that configuring ConQuest's flow definition [19] for a single subnet context, such as an IP with a /24 subnet mask, is feasible. However, the programmable switch's resource constraints prevent it from accommodating multiple subnet contexts simultaneously.

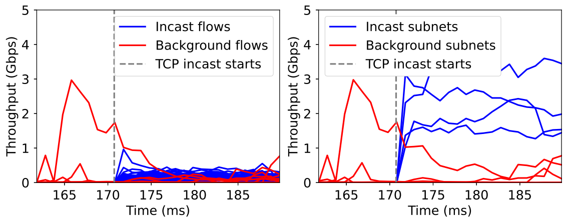

Fig. 2. Per-flow throughput and aggregated per-subnet throughput during a microburst.

Therefore, it is necessary to analyze *across flows* to pinpoint the group of bursty flows together and determine the root cause of the incast. Recent works like Precision [16, 17] also show the importance of across-flow analysis in detecting DDoS attacks. The hierarchical heavy hitter detection algorithms can achieve higher accuracy with an increasing number of counters (more memory usage). To diagnose network problems in Table 1, datacenter operators usually need to analyze group-level flow behaviors at different dimensions (flow, port, priority, or other metadata). With an increasing number of different grouping levels, it requires more memory capacity to support across-flow analysis. Programmable switch only provides tens of MB on-chip memory per pipeline. Modern FPGA card offers several gigabytes on-chip high bandwidth memory (HBM) [9], which provides $\approx 100ns$ access latency and up to 460GB/s bandwidth [77]. Thus, FPGAs enable much more accurate contextual analysis with larger hierarchies.

Comprehensive analysis. Many queries, e.g., as listed in Table 1, involve calculating *composite functions* (change detection, correlation, clustering) over the input data. The complexity far exceeds what traditional programmable switches can offer, and warrants the more powerful and flexible computational capability of FPGAs. One such case is analyzing *metastable failures* [7], a phenomenon caused by the interaction of application-layer connection pool logic and the underlying network-layer load balancing. In this incident reported by Meta, and solved after more than two years, a Most-Recently-Used (MRU) connection pool to the database backend was used with flow-based Equal-Cost Multi-Path (ECMP) load balancing. Although MRU can minimize the number of unused connections compared with other reusing policies (e.g., LRU), the system enters a delicate unstable state, where any perturbation will push the system towards the imbalanced failure state [7]. Here, any packet loss will cause a connection to finish slower, which puts it on top of the MRU connection pool to be used again immediately. When a link is particularly congested, and lots of flows are suffering from congestion or losses, the MRU policy may end up only using this link. Traditional methods can only detect ECMP imbalance when the aggregated throughput differs across different links, at which point it is too late to avoid entering a metastable failure state. However, it is possible to detect the MRU pool earlier by calculating *comprehensive analysis functions* such as those required for causal analysis. Using Cross Correlation [2], we can identify the relationship between a packet loss or ECN congestion tag and the connection being subsequently reused. Once causality is confirmed, we can take remedial actions to avoid the system entering the metastable failure state.

2.2 The Need for Performance.

Low latency: Several examples in Table 1 (microburst, LDoS attack, Slowloris Attack) can benefit from a fast and immediate reaction to performance anomaly events, which can greatly reduce application performance impact.

To illustrate this point, we create an *ns-3* simulation that sends background flows to a receiver with a single 10Gbps link at 90% utilization. Each flow uses DCTCP and the flow size is randomly drawn from the DCTCP [14] distribution. Similarly to the incast setting in HPCC [51], we then send 60 incast flows (each of size 500KB) simultaneously from 60 different senders to the receiver. The total incast traffic is 30MB and it takes 24 ms to finish at 10Gbps link. We measure the flow completion time (FCT) of background flows starting in this time window and calculate the relative slowdown compared to without incast case. The reaction we perform in this example is to reroute incast flows to another low utilization path, and *reaction time* is defined as the time interval between incast happening and when the reaction is performed.

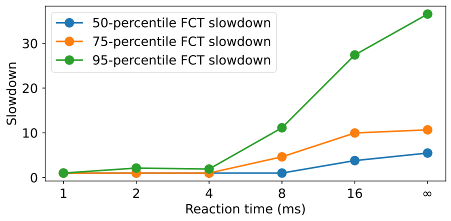

Fig. 3. Background flows FCT slowdown under different reaction times.

Figure 3 shows the background flow FCT slowdown under different reaction times. The data point with an x-value of infinite reaction time represents the case with no reaction, resulting in a 5.5×, 10.7×, and 36.5× slowdown for the 50-, 75-, and 95-percentiles FCT of the background flows. A fast 1ms reaction time mitigates most of the impact, translating to only 1.9× slowdown of 95-percentile FCT of background flows and zero slowdowns for 50- and 75-percentiles. With 16ms reaction time, background flows still experience severe 3.8×, 4.6× and 27.4× FCT slowdowns at 50-, 75- and 95-percentiles.

While many previous monitoring systems [30, 59] use end-host CPUs to aid the monitoring analysis, we believe it would be better to execute analysis on computing units near the switch dataplane to provide the possibility of fast reaction. In the case study of Sonata, it requires up to 11 seconds for end-host CPUs to analyze traffic originating from the Tofino switch to detect Zerror attacks [30]. Using FPGA as a co-processor of the switch can leverage its low latency processing [48] advantage to enable fast reaction.

High throughput: Switch dataplanes are augmented with a control-plane CPU to perform functions that can not be done in the dataplane [80]. Due to power constraints, the switch CPU is less powerful and, by design, is handling control tasks instead of dataplane traffic. Moreover, the switch's CPU is needed for various other functionalities such as communicating with a controller [49] or installing rules [44], which are needed for policy changes [57], traffic reroute [11].

2.3 FPGAs to the Rescue

To summarize, the ASIC pipeline of programmable switches can provide performance, but it has limited support for comprehensive analysis (§2.1). In contrast, relying on either collectors or switch CPUs can help with the analysis but does not meet our requirements on performance (§2.2).

FPGAs meet both requirements: they are high-performance architectures able to process 100Gbps network traffic at dataplane speeds [91]. They are more energy efficient than CPUs [83] and allow more comprehensive analysis than switch ASICs, as generic as CPUs. Finally, their partial reconfiguration feature allows flexibly adding queries without downtime in the monitoring applications.

Recognizing the benefits of FPGAs, switch vendors are including them in products, connected through high-bandwidth buses to the ASIC pipeline. For example, the Intel Tofino Expandable Architecture is being used by multiple companies for a variety of applications, such as NAT translation, firewalls, tunnel termination, and layer-4 load balancing [3].

3 F3 Design

A high-level overview of our design is shown in Figure 4. After a pre-deployment phase, each query is partitioned between the FPGA and switch (§ 3.1), and we apply cross-query level optimizations (§ 3.2). We ensure that the deployment meets the throughput of the FPGA (§3.3) and finally, we analyze the result and report them to the operator (§3.4).

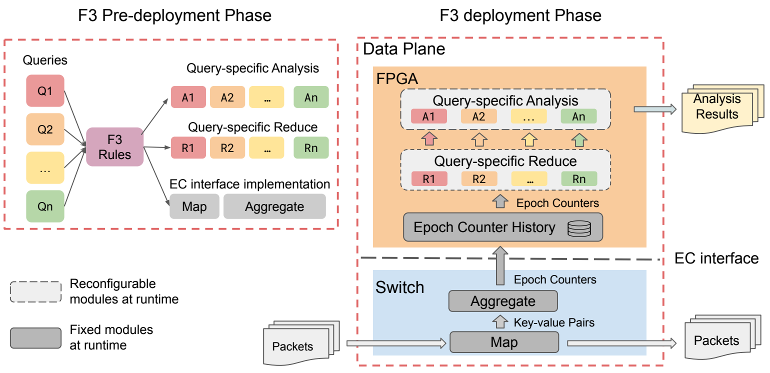

Fig. 4. The F3 framework.

3.1 Labor division for single query

The FPGA can run comprehensive analysis (e.g., analysis functions in Table 1) but can not sustain the Tbps-scale processing of switches. In contrast, switches are more limited in processing. Also, their limited resources need to be used for many other functions, such as ACL, load balancing, and potentially assisted congestion control (e.g., HPCC [51]). Here we want to explore the right division of labor between switches and FPGA for just a single query. The goal is to satisfy the FPGA's throughput constraint while minimizing switch resource usage. We then discuss optimizing this labor division strategy for multiple query cases (§3.2).

We observe that many monitoring queries already have *throughput reduction operators* that decrease the data stream size [30, 59]. These operators include *filter* (which drops data that meets given predicates), *sample* (which randomly selects data at a given ratio), and *aggregate* (which applies aggregation function such as `sum`, `max`, `count` to data of the same key). Listing 1 is the Slowloris attacks detection query [30], where network operators identify hosts that open many low-traffic volume TCP connections. This query has in total six throughput reduction operators (three aggregate operators and three filters).

We propose to run a few throughput reduction operators at the switch to decrease the amount of telemetry data transmitted to FPGA. To determine the appropriate number of each type of

Listing 1. Slowloris Attacks Detection

```
n_conns = packetStream # Throughput: 3.2Tbps
 .filter(p => p.proto == TCP)}
 .aggregate(keys=(dIP,sIP,tcp.sPort), val=None,
           f=distinct) # Throughput: 14Gbps
 .aggregate(keys=(dIP,), val=None,
           f=count)  # Throughput: 4Gbps
n_bytes = packetStream
 .filter(p => p.proto == TCP)}
 .aggregate(keys =(dIP,), val=ipv4.totalLen, f=sum)}
Slowloris = n_conns.join(keys=(dIP,), n_bytes)
 .map((dIP, (con,byte)) => (dIP, (con/byte)))
 .filter((dIP, con/byte) => (con/byte > Th2))
```

Listing 2. Slowloris Attacks Detection (reduce-first)

```
reduce_prefix = packetStream # Throughput: 3.2Tbps
 .filter(p => p.proto == TCP)
 .aggregate(keys=(dIP,sIP,tcp.sPort),
           val=ipv4.totalLen,f=sum) # Throughput: 14Gbps
n_conns = reduce_prefix
 .aggregate(keys=(dIP,sIP,tcp.sPort), val=None,
           f=distinct)
 .aggregate(keys=(dIP,), val=None, f=count)
n_bytes = reduce_prefix
 .aggregate(keys =(dIP,), val=ipv4.totalLen, f=sum)
Slowloris = n_conns.join(keys=(dIP,), n_bytes))
 .map((dIP, (con,byte)) => (dIP, (con/byte)))
 .filter((dIP, con/byte) => (con/byte > Th2))
```

throughput reduction operator, we need to consider both FPGA's throughput constraint and switch resource usage. First, the telemetry data stream after the first aggregate operator in the monitoring query has the throughput of ~10 Gbps for Tbps-level network traffic, which already satisfies the FPGA throughput constraint. Second, aggregate needs much more switch resources than filter and sample. Aggregate is a stateful operator that needs switch SRAM resource to implement the hash table [30], while filter and sample only require a single match-action table on the switch. Allocating more aggregate operators on the switch can considerably increase switch resource usage.

As a result, we determine that the appropriate allocation for the throughput reduction operators on the switch is up to the first aggregate operator in the query, which we refer to as the *reduction operator prefix (ROP)* of the query. The ROP starts with zero or multiple filter/sample operators and ends with one aggregate operator. We investigated common monitoring tasks as shown in Table 2 and the core insight here is monitoring query can be written in a *reduction-first* way, where the beginning of the query is reduction operator prefix. We take the query for Slowloris attack detection as an example (Listing 1) and this query can be rewritten in a reduction-first way. The start of the query (Listing 2) consists of filter and aggregate operators, both of which are types of reduce operators. Table 2 shows more queries that can be written as reduction-first queries.

Given a reduction-first query, we allocate its ROP to the switch and the remaining part of the query (including its analysis function) to the FPGA. This approach has two benefits: firstly, the telemetry data transmitted from the switch can satisfy FPGA's throughput constraint. Second, ROP has minimal switch resource usage since it has only one aggregate operator.

3.2 Optimization for multiple queries

To perform labor division for n reduction-first queries between the switch and FPGA, one strawman solution is to allocate n ROPs on the switch and the remaining parts of n queries to the FPGA. However, increasing the number of ROPs deployed on the switch may lead to FPGA throughput constraint violation and high switch resource usage. So we propose an optimization to reduce the number of ROPs allocated on the switch, which we refer to as *ROP merging*.

The intuition of ROP merging is to find a new ROP R such that the output of ROPs before merging (e.g., $\{R_1, R_2, .., R_n\}$) can be derived from R's output. As a result, we can only allocate R on the switch instead of allocating n ROPs on the switch. To simplify the discussion, we only discuss ROP merging for two ROPs R_1 and R_2 (but it can be generalized to multiple ROPs). Since ROP ends with an aggregate operator, the output of ROP is `(key, value)` pairs after each query-defined time interval or window. We use $\{(k_1, v_1)\}$ to designate the key-value pairs from R_1 and similarly $\{(k_2, v_2)\}$ for R_2. k_1 and k_2 represent the aggregation keys in R_1 and R_2. The target ROP R after merging outputs $\{(k, v)\}$, which we can apply extra throughput reduction operators to get $\{(k_1, v_1)\}$ and $\{(k_2, v_2)\}$. Here, we discuss the merging opportunity based on the relationship between k_1 and k_2.

Case 1: k_1 and k_2 has subset relationship. Here, we assume k_1 is a subset of k_2 for discussion purpose. There are merging opportunities for the following two sub-cases.

(a) When R_1 and R_2 have the same filter and sample operators, we can use R_2 as the merged ROP. For example, we have R_1 and R_2 as defined below. And R_2 can be used as the merged ROP because we can apply extra `aggregate(keys=(sIP))` to derive R_1's output from R_2's output.

Case 1 (subset): ROP merging example with same filters

R_1=**filter**(tcp.flags==SYN).**aggregate**(keys=(sIP))
$\wedge$ R_2=**filter**(tcp.flags==SYN).**aggregate**(keys=(sIP,dIP))
$\Rightarrow$ $R = R_2$

(b) When R_1 and R_2 have different filter and sample operators, but the header fields used in these filter/sample predicates are a subset of k_2, we can get merged ROP R in two steps. First, we perform the disjunction of predicates inside R_1 and R_2 for filter and sample operators separately. We then use them as filters or sample predicates inside R. Second, we use R_2's aggregate operator as the aggregate operator in R. In the below example, we perform disjunction for predicates `dport > 80` and `dport < 60` and use R_2's aggregate operator in R.

Case 1 (subset): ROP merging example with different filters

```
R1=filter(dPort>80).aggregate(keys=(sIP))
∧ R2=filter(dPort<60).aggregate(keys=(sIP,dPort))
⇒ R=filter(dPort>80||dPort<60).aggregate(keys=(sIP,dPort))
```

We choose to do merging for the above two sub-cases when k_1 and k_2 have a subset relationship. The merged ROP R can save switch resource usage because the switch only needs to support one aggregate operator (instead of two for when deploying R_1 and R_1). It can also reduce telemetry data throughput sent to FPGA.

Case 2: k_1 and k_2 are not subset of each other. In this case, merging ROPs may increase the switch resource usage and telemetry data throughput. In the below example, we can union k_1 and k_2 to get the merged ROP. However, the number of distinct `(sIP, dIP)` pairs after each aggregation interval can be much larger than the sum of distinct `sIP` and distinct `dIP`. As a result, the merged ROP R needs more switch resources to allocate a larger hash table and also send more telemetry data. So we choose to not merge for this case.

Case 2 (not-subset): ROP merging risk example

```
R1=aggregate(keys=(sIP)) ∧ R2=aggregate(keys=(dIP))
⇒ R=aggregate(keys=(sIP,dIP))
```

In summary, ROP merging can reduce switch resource usage and telemetry data throughput when deploying multiple queries. But ROP merging can not guarantee satisfying FPGA throughput constraints when network operators want to deploy a large number of queries or there is a traffic change in the runtime. In the next subsection (§3.3), we will introduce safeguards to meet FPGA throughput requirements.

3.3 Safeguard for Meeting the Throughput of FPGA

There are no fundamental constraints on the complexity of queries that can be supported by the FPGA, which is as flexible as high-end CPUs [79]. For instance, prior work such as N3IC [68] demonstrates that FPGAs can execute machine learning models on line-rate network traffic. However, when a large number of queries or highly resource-intensive queries are deployed simultaneously, the FPGA may struggle to handle the monitoring traffic from the switch at full line rate (e.g., 100Gbps). To prevent queries from exceeding the FPGA's throughput capabilities, F3 employs both compile-time and run-time safeguards, ensuring that queries remain within the FPGA's throughput limits.

Compile-time. Using training data in the form of representative packet traces provided by operators, F3 runs traces to estimate the aggregated bandwidth consumed by ROPs. If the estimated bandwidth is smaller than FPGA's throughput constraint `max_bw`, F3 accepts this set of ROPs. Otherwise, F3 needs users to rewrite their reduce-first queries such that ROPs can perform further throughput reductions or reduce the number of reduce-first queries to be deployed. F3 then repeats the previous steps to optimize these ROPs. This workflow is similar to the profile-guided techniques which collect performance profiles to optimize software in the production environment (e.g., Facebook Bolt [60] and Google AutoFDO [18]). The assumption of having representative historical traces is valid, as prior work, such as Sonata [29], also uses them to optimize monitoring query planning.

Run-time. During a transient period where current traffic patterns deviate from historical traces, if the output counter stream exceeds `max_bw` during runtime, F3 automatically downscales the counter stream to reduce the bandwidth consumption. That is, if the `max_bw` allows us to stream X counters, and in the current epoch, we observed Y distinct keys, a uniformly random subset of X counters will be sent. This approach is equivalent to flow sampling when the key is a 5-tuple. As demonstrated in prior work [15], many monitoring queries exhibit accuracy that degrades gracefully based on the flow sampling probability, $p = \min\{1, X/Y\}$. For example, consider detecting super spreaders or port scans (srcIPs that send packets to many dstIPs or many ports on the same destination) in Table 2. Prior works (e.g., Sonata [30]) have considered a fixed threshold T (e.g., 1000) such that any srcIP that communicates with more than that many dstIPs/ports is considered malicious. After sampling, we check which srcIPs communicate with more than $T \cdot p$ destinations. As long as p is not exceedingly small, concentration bounds imply that F3 would be able to satisfactorily identify the culprits. Namely, if the srcIP communicated with Z destinations, the number of sampled counters with high probability be $Z \pm \Theta(\sqrt{Z \cdot (1-p) \cdot p^{-1}})$ (Θ function represents asymptotic tight bounds). For example, with $p = 10\%, T = 1000$, with probability 99% we will flag a srcIp that communicates with at least 1500 dstIps but will not flag one that communicates with at most 670.

3.4 Telemetry data analysis at FPGA

F3 FPGA is composed of three modules: *epoch-counter history*, *query-specific reduce*, and *query-specific analysis*. The epoch-counter history module is a fixed module at runtime, which keeps epoch counters output from ROPs on the F3 switch in recent history. Operators can write their query-specific reduction and analysis modules using either RTL language (e.g., Verilog) or High-level Synthesis (e.g., Vitis HLS [10]). They can also reconfigure the reduce and analysis modules at runtime based on the queries. With partial reconfiguration, operators can introduce new queries without impacting current queries deployed on F3 FPGA.

Epoch Counter History The FPGA maintains epoch counter history in a circular buffer. It enables temporal analysis to find the pattern across different epochs, which is necessary for analyzing examples like *microburst analysis* and *detect LDoS attack* in Table 1.

With FPGA's large on-chip memory, the epoch-counter history module can maintain many epochs. For example, 1024 epochs only take 1.488% FPGA on-chip BRAM (see Section 5.2). Epoch-counter history enables temporal analysis in the query-specify reduction and analysis stage.

Query-specific Reduce. With merging rules in section 3.2, queries with ROPs merged under merging rules take the same epoch counters as input. But we need query-specific reduction to break these epoch counters into query-specific inputs. For example, detect super spreader in Table 2 needs to apply `aggregate(keys=(sIP,dIP),f=distinct)` to break epoch counters output by `R=aggregate(keys=fiveTuple,f=distinct)` into unique IP pairs. Operators can implement one reduce layer for each of these merged queries. F3 parallelizes these reduction layers in FPGA.

These reduction layers can leverage FPGA's large memory to minimize the error. For example, the aggregation operator can implement larger hash tables at FPGAs than at switches, which reduces hash collision.

Example	Query code

Super spreader [81]	`reduce_prefix = packetStream.aggregate(keys=(sIP,dIP),val=None, f=distinct)`
	`analysis = reduce_prefix.aggregate(keys=(sIP,), val=None, f=count).filter((sIP, cnt) => cnt > Th)`

Port scan [37]	`reduce_prefix = packetStream.aggregate(keys=(sIP,tcp.dport),val=None, f=distinct)`
	`analysis = reduce_prefix.aggregate(keys=(sIP,), val=None, f=count).filter((srcIP, cnt) => cnt > Th)`

TCP Incomplete Flows [82]	`reduce_prefix = packetStream.aggregate(keys=(sIP,tcp.flags), val=None, f=count)`
	`n_syn = reduce_prefix.filter((sIP,tcp_flags),cnt) => (tcp_flags == SYN).map(((sIP, tcp_flags),cnt)=>(sIP, cnt))`
	`n_fin = reduce_prefix.filter((sIP,tcp_flags),cnt) => (tcp_flags == FIN).map(((sIP, tcp_flags),cnt)=> (sIP,cnt))`
	`analysis = n_syn.join(keys=(sIP,), n_fin).map((sIP,(cnt1,cnt2)) => (sIP,cnt1-cnt2)).filter((sIP,cnt) => cnt>thr)`

Newly opened TCP Conns [82]	`reduce_prefix = packetStream.aggregate(keys=(dIP,tcp.flags), f=count)`
	`analysis = reduce_prefix.filter(((dIP,tcp_flags),cnt)=>tcp_flags==SYN)`
	`                        .map(((dIP,tcp_flags), cnt)=>(dIP,cnt)).filter((dIP,cnt)=>cnt>thr)`

Slowloris attack detection [30]	`reduce_prefix = packetStream.filter(p => p.proto == TCP).aggregate(keys=(dIP,sIP,tcp.sPort), val=ipv4.totalLen, f=sum)`
	`n_conns = reduce_prefix.aggregate(keys=(dIP,sIP,tcp.sPort), val=None, f=distinct).`
	`                        aggregate(keys=(dIP,),val=None,f=count)`
	`n_bytes = reduce_prefix.aggregate(keys =(dIP,), val=ipv4.totalLen, f=sum)`
	`analysis = n_conns.join(keys=(dIP,), n_bytes)).map((dIP,(con,byte))=>(dIP, (con/byte))).`
	`          filter((dIP,con/byte)=>(con/byte>Th2))`

LDoS attack detection [45, 92]	`reduce_prefix = packetStream.aggregate(keys=5tuple, val=ipv4.totalLen, f=sum)`
	`analysis = reduce_prefix .autoCorrelation(thr, nEpochs=n)`

Microburst analysis [19, 38, 76]	`reduce_prefix = packetStream.aggregate(keys=5tuple, val=ipv4.totalLen, f=sum)`
	`ip_prefix_bytes = reduce_prefix.map((5tuple,bytes)=>(sIP/24,bytes).aggregate(keys=(sIP/24,),val=bytes,f=sum)`
	`analysis=ip_prefix_bytes.changeDetection(threshold=Th1, tA=n1, tB=n2)`
	`    .join(key=(sIP/24,), reduce_prefix).filter((5tuple, nbytes) => nbytes > Th2).`
	`                 map((5tuple, _) => (sIP/24, nbytes))`

Pulse-wave DDoS Analysis [12]	`reduce_prefix = packetStream.aggregate(keys=5tuple, val=None, f=count)`
	`analysis = reduce_prefix.clustering()`

Priority flow contention [72, 73, 76]	`reduce_prefix = packetStream.aggregate(keys=(priority, 5tuple), val=ipv4.totalLen, f=sum)`
	`priority_bytes = reduce_prefix.aggregate(keys=(priority,), val=ipv4.totalLen,f=sum)`
	`flow_priority = reduce_prefix.aggregate(keys=(priority, 5tuple), val=None, f=distinct)`
	`analysis = priority_bytes.join(keys=(priority,),flow_priority).`
	`                          filter((priority,(n_bytes,5tuple))=>(priority<Th1)&&(n_bytes>Th2))`

Packet Loss detection [50]	`reduce_prefix = packetStream.aggregate(keys=5tuple, val=None, f=count)`
	`analysis = reduce_prefix.flowLossDetection(thr, nEpochs=n)`

Table 2. Reduce-first Query Table.

Query-specific Analysis. Query-specific analysis module provides the core analysis part of each query. For example, in the second column of Table 2, the query code not colored in blue is the query-specific analysis part. Here, we use two examples to show that FPGA can support advanced analysis functions.

Change detection. Change detection can capture significant changes in short-term behavior [42]. The microburst query (Table 1) uses change detection to detect the bursty flows or sub-networks in a short time window. Existing works like ConQuest use simple thresholding to flow's weight in the queuing buffer to detect bursty flows, which could incorrectly report large background flows as bursty flows. Change detection can distinguish large background flows from real bursty flows by analyzing temporal change patterns. We present one version of the change detection function based on previous work [53, 54, 65].

Consider two adjacent time intervals t_A and t_B epochs. If the volume for a flow x in interval t_A is $S_A[x]$ and $S_B[x]$ over t_B, we define the change of a flow x as the volume differences in the two time intervals: $D[x] = |S_A[x] - S_B[x]|$. We define a flow as a *heavy* change flow if the difference in its signal exceeds ϕ percentage of the total changes over all keys. The total changes are defined as $D = \sum_{x \in [n]} D[x]$.

Previous switch-based change detection needs control-plane involvement and only performs simple change detection when t_A and t_B are the same as single epoch length [54]. This is due to the limited number of stages on the programmable switch. FPGA can provide change detection across multiple epochs fully on the data plane.

Flow-level packet loss detection. The detect packet loss query (Table 1) detects flow-level packet loss by comparing the per-flow packet counters from upstream and downstream switches. In F3, FPGA is directly connected to one switch (local switch) and it can receive epoch counters from the local switch and remote switches through packet forwarding. We use k-set [24] to aggregate epoch counters `(flow,n_pkts)` from different switches. The k-set is a dictionary data structure that supports both addition and subtraction of a sequence of `(key,value)` pairs. k-set also provides an operation `RecoverSet` that can return all `(key,value)` pairs with non-zero value inside the k-set. k-set is a randomized data structure and takes a confidence parameter δ. It can provide $1 - \delta$ probability guarantee that `RecoverSet` can succeed when the number of satisfied keys is less than k.

To detect flow-level packet loss, k-set adds `(flow,n_pkts)` counters from upstream switches and subtracts `(flow,n_pkts)` counters from downstream switches. `RecoverSet` operation can return flow-level loss counters `(flow,n_loss_pkts)`. Compared with other dictionary data structures (e.g., a hash table), the k-set has much smaller memory usage, which is determined by the number of lossy flows instead of the total number of flows. However, k-set is resource-consuming to implement on the switch (costs lots of ALUs to simulate multiplication operations), while FPGA can easily support it.

4 Implementation

We have implemented F3 on a Tofino 32D switch ASIC [4] connected to Xilinx/AMD Alveo U250 FPGA card [1] to emulate an FPGA co-processor. The F3 switch code, consisting of the Map and Aggregate modules, is written in P4 while the FPGA code, including the Epoch counter history, Query-specific reduce and Query-specific analysis modules, is written in Verilog.

F3 Switch There are two types of modules at the switch:

Map. We extract fields from packet headers and evaluate predicates, which are boolean operations on packet header fields (e.g., `dport > 80`) or fields stored in user metadata. The resulting flags from these evaluations are then stored in the user metadata.

Aggregate. We use the available registers to implement a hash table for the epoch-counter (EC). At the end of each epoch, we use Tofino's data plane packet generator to export the counter to the FPGA, creating a batch of packets, each recirculating several times to minimize the header overheads. Since the data plane packet generator is hardware-based, users can configure the counter readout interval with a fine-grained period of $100\mu s$ or even lower and reach 100Gbps throughput.

F3 FPGA There are three types of modules in the FPGA:

Epoch counter history. Each counter-history is implemented using FPGA's BRAM. We declare registers in Vitas HLS [10] and Vitas can find the best BRAM layout on FPGA.

Query-specific Reduce. For the aggregate operator, we use a hash table which is implemented using FPGA's BRAM. The hash function is an efficient *multiply-shift* function based on [74]. For other reduce operators (filter and sample), they are directly expressed in Vitas HLS.

Query-specific Analysis. Here, we focus on discussing the implementation of advanced functions mentioned in Section 3.4. (1) *Change Detection.* The change detection module has a change table implemented in BRAM. First, we calculate the change for each flow and put it into the change table. Second, we calculate the sum of all absolute changes. Third, we check if the change of each flow is larger than a certain threshold of the total change. Since FPGA is more efficient in performing integer arithmetic operations, we use integer multiplication to approximate the checking condition $\lceil \frac{1}{\phi} \rceil \cdot D[x] \geq D$. (2) *Flow-level packet loss detection.* We implement the k-set data structure in Verilog. The two-dimensional hash table is implemented using BRAM. Since `recoverSet` operation needs integer division, we translate division into multiplication and bit operation, which is an optimization technique commonly used in modern compilers [78].

5 Evaluation

We now show the benefits of F3 with FPGA as a coprocessor for network telemetry: F3 can support a variety of queries with comprehensive analysis with high performance and low resource usage.

5.1 Supporting Comprehensive Analysis

We demonstrate the flexibility of F3 with two examples *Pulse-wave DDoS Analysis* and *Microburst Analysis*. For each example, we compare F3 with two alternative solutions in simulation : state-of-the-art switch dataplane monitoring systems and the ideal CPU-based solution that runs comprehensive analysis algorithms at line rate.

5.1.1 Pulse-wave DDoS Analysis. Pulse-wave DDoS attacks are a new volumetric attack type formed by short, high-rate traffic pulses [12]. ACC-Turbo [12] introduces an online clustering algorithm to characterize pulse-wave DDoS attack patterns identify the cluster for each incoming packet, and perform real-time reactions (e.g., programmable packet scheduling) for malicious flows.

Experiment setup. We leverage the open-source simulation code from ACC-Turbo [27] , which simulates a switch processing network traffic. And we feed CICDDoS-2019 [66] trace, as used in ACC-Turbo [12], at 108x speed (simulating 100Gbps network load), which consists of a series of DDoS attacks.

Alternative solutions. We compare three clustering algorithms: (1) *ACC-Turbo on the switch*: We use the *ACC-Turbo* algorithm takes *srcip* and *dstip*, *sport*, *dport*, *ttl*, and *pktlen* and follow the approach in ACC-Turbo [12] that uses each byte of them as features. The algorithm runs four clusters as suggested in [12] to fit in the limited stages and ALUs of a Tofino switch. (2) *Online K-means on F3*: F3 takes the input of epoch counters every $100\mu s$, and runs query-specific reduce to extract each byte of the *srcip* and *dstip*, *sport*, *dport* as features. Based on the packet counts in ECs, the FPGA generates these features for each packet to feed into the clustering algorithm. Note that we do not use features *ttl*, and *pktlen* because they are per-packet information that epoch counters cannot collect. F3 runs the online k-means solution with a constant update ratio of $\frac{1}{16}$ [6] to make it feasible to implement on FPGA. (3) *Offline K-means on the ideal CPU* We assume an ideal CPU can take all the per-packet features (same as (1)) at line rate and run the best offline K-means solution [41]. For (2) and (3), we vary the number of clusters from 4 to 20.

We use two metrics: purity and recall. Same as [12], we label each cluster as either majority-benign or majority-malicious. Purity measures the number of packets matching their cluster label divided by the total number of packets. Recall measures the percentage of malicious packets clustered to the majority-malicious clusters.

Results. Figure 5 shows that F3 improves the purity of *ACC-Turbo on the switch* by up to 11.7% and recall by up to 35.9%. This is because F3 supports more comprehensive analysis than switches can support. Compared to the ideal CPU, F3 only loses 2% purity and 8.0% recall. This is because F3 uses fewer features and runs the approximate solution of online K-means. With more clusters, the purity and recall gap between F3 and the ideal CPU becomes smaller because more clusters can make both the purity and recall closer to optimal.

5.1.2 Microburst Analysis: Microburst is common in data centers [87] and operators are interested in understanding the interplay with application workloads and burst properties [25]. One of the most important steps is finding the microburst's contributing flows.

Experiment setup:

Topology. We use the ns-3 [5] simulator to test a topology with 8 spine and 8 leaf switches with 128 hosts connected by 10Gbps links with $2\mu s$ propagation delay [84]. We set the switch queue size as 700KB. We use DCTCP [14] as the default congestion control algorithm.

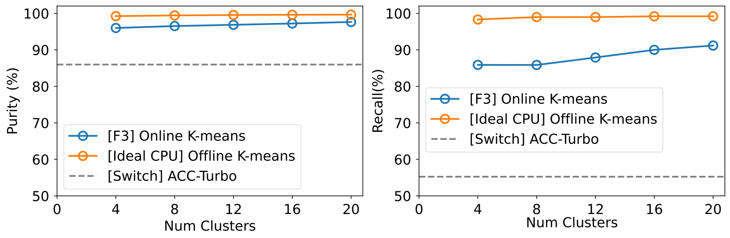

Fig. 5. DDoS clustering purity and recall

Traffic. We feed background traffic based on *DCTCP_CDF* [14] with average 60% network load. Like in [51], we generate incast traffic by randomly selecting 60 senders and one receiver, each sending one flow of 500KB. The incast traffic load is 2% of the network capacity. During each incast, a microburst happens at the leaf switch port that connects to the receiver. These incast flows are the ground truth contributing flows for the microburst. We feed the packet traffic received at each leaf switch port to the various solutions below to detect the contributing flows of the microburst. The simulation lasts 200ms and there are 64 incasts during the simulation in total.

Alternative solutions: We compare three microburst analysis algorithms: (1) *ConQuest on the switch[19]: ConQuest* is the state-of-the-art solution for microburst analysis on switches. We configure the snapshot and count-min sketch parameters as the ConQuest paper suggested. We varied its queue occupancy threshold from 1% to 10%. (2) *Change Detection on F3*: In F3, the switch exports epoch counters every 100μs. The FPGA runs change detection as discussed in Section 4. To fit the FPGA architecture, change detection has two approximations: First, we use a hash table to aggregate changes for each flow which may incur collision; second, we use integer arithmetic to replace floating point arithmetic. (3) *Change Detection on the ideal CPU*: For the ideal CPU case, we run the change detection algorithm without any approximation: We run an ideal hash table without collisions and use floating point arithmetic. For (2) and (3), we configure t_A and t_B as 8 epochs and vary the change detection threshold from 4% to 40%.

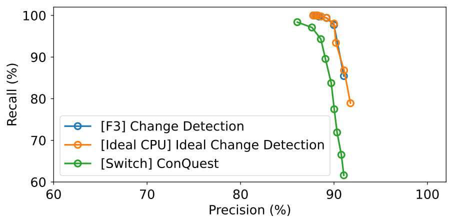

Fig. 6. Microburst analysis precision-recall curve

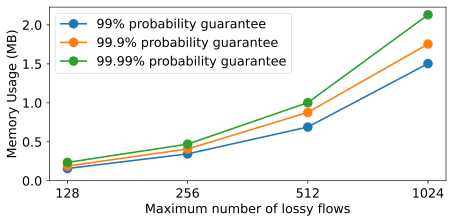

Fig. 7. k-set memory usage

Results. Figure 6 shows the precision-recall curve of the three microburst analysis solutions. F3 has a better precision-recall tradeoff than ConQuest on the switch. F3 can improve the recall by up to 24% for the same precision rate compared with ConQuest. Compared with the ideal CPU, F3's precision-recall curve is close to the ideal CPU. For example, F3's precision is only 0.7% smaller than the ideal CPU when both achieve 98.0% recall. The difference is caused by the two approximations in the FPGA implementation of change detection we mentioned before.

5.2 High Performance

To demonstrate the generality of the F3 framework, we implemented ten telemetry queries in Table 5 on Xilinx U250 FPGA [1] using Vitis HLS 2020.2 [10].

All queries can be synthesized at 340MHz frequency, which means FPGA can process 100Gbps throughput. FPGA can receive up to 781 million epoch counters per second (each epoch counter is 16 bytes).

F3 runs three stages at the FPGA: epoch-counter history, query-specific reduction, and query-specific analysis. Due to pipelining, F3 FPGA's end-to-end analysis latency is determined by the bottleneck stage. When we run advanced query-specific analysis (such as correlation and change detection), the end-to-end latency is determined by the analysis latency.

Table 3 shows the latency of a few advanced query-specific analysis functions used in our queries. The correlation latency is the time it takes to calculate the auto-correlation score across 512 history epochs for one flow. The change detection latency is the time it takes to find heavy change flows across 128 different flows (t_A = 8 epochs and t_B = 8 epochs). The flow-level loss detection latency is the time it takes to get flow loss by performing `RecoverSet` operation to k-set (k = 128).

Analysis functions	**Latency** (*ns*)
Correlation	791
Change detection	6051
Flow-level loss detection	6124

Table 3. Latency involving advanced query-specific analysis.

	Stages	ALUs	SRAM
F3 for all queries	4	20.8%	2.1%
ACC-Turbo for Pulse-wave DDoS	12	72.9%	15.4%
ConQuest for microburst analysis	9	18.8%	3.3%

Table 4. Switch resource usage

Our FPGA can finish all query-specific analysis functions in Table 3 less than 6.2 μs. In contrast, the CPU's median packet forwarding latency using DPDK without analysis is already higher (7.8 μs reported in [22]). The change detection and flow-level loss detection take more time than correlation because they both have more memory access across epochs and flows. The loss detection analysis also needs complex computation in the k-set recovery stage.

5.3 Resource usage

We measure the resource usage of F3 at both the switch and the FPGA.

Switch: Table 4 shows that F3 uses only four stages, 20.8% of ALUs, and 2.1% of SRAM to support all ten queries.

In contrast, the switch implementations for specific queries can take more resources. (Note that some queries do not even have a switch-only solution.) We take ACC-Turbo on the switch for Pulse-wave DDoS analysis and ConQuest for microburst analysis as an example. ACC-Turbo on the switch takes 12 stages to maintain the states for four clusters and 72.9% ALUs to run computation for online clustering. ConQuest also takes more stages and SRAM to maintain multiple snapshots history on the switch. F3 has 2.0% higher ALU usage than ConQuest because F3 needs more computation to generate epoch counters that work for all queries.

FPGA: Overall, it takes around 26.9% of BRAM and 6.1% of lookup tables to support all ten queries together. We also breakdown the resource usage into three stages:

For the epoch-counter history module, we implement one that keeps 1024 epochs and each epoch is a hash table of 2^{16} rows. The key of the epoch counter is 128-bit and the value of the epoch counter is 32-bit. The epoch-counter history module uses just 1.448% of total BRAM on FPGA.

Table 5 shows the resource usage of individual queries for the query-specific reduction and analysis modules. Port scan query only costs less than 0.9% BRAM and less than 0.03% for other resources because it only has a few basic dataflow operators (aggregate and filter). For the packet loss detection query, we set $k = 128$ which means the k-set can capture a maximum number of 128 lossy flows. We configure k-set to take `5-tuple` (104-bit) as key and packet counter (16-bit) as value to get flow-level packet loss counter. This query costs around 8.8% BRAM, 6.1% DSP, 1.1% FF, and 1.1 % LUT. The reason is that the k-set needs to maintain two-dimensional tables and do multiplication for loss detection.

	Query-specific Reduce				Query-specific Analysis			
Query	BRAM	DSP	FF	LUT	BRAM	DSP	FF	LUT
Super spreader [81]	0.595%	0.041%	0.025%	0.020%	0.595%	0.000%	0.009%	0.012%
Port scan [37]	0.595%	0.024%	0.021%	0.015%	0.298%	0.000%	0.001%	0.006%
TCP Incomplete Flows [82]	0.893%	0.008%	0.011%	0.017%	0.595%	0.000%	0.011%	0.016%
Newly opened TCP Conns [82]	0.893%	0.008%	0.011%	0.017%	0.595%	0.000%	0.010%	0.012%
Detect slowloris attack [30]	1.488%	0.106%	0.026%	0.021%	0.893%	0.000%	0.012%	0.018%
Detect LDoS attack [45, 92]	1.488%	0.179%	0.027%	0.021%	1.488%	0.179%	2.679%	2.443%
Microburst analysis [19, 38, 76]	1.488%	0.179%	0.027%	0.021%	1.786%	0.179%	1.760%	1.984%
Pulse-wave DDoS Analysis [12]	1.488%	0.179%	0.027%	0.021%	0.000%	0.065%	0.036%	0.485%
Priority flow contention [72, 73, 76]	1.488%	0.244%	0.028%	0.021%	1.488%	0.244%	0.033%	0.024%
Detect packet loss [50]	1.488%	0.179%	0.027%	0.021%	7.328%	5.982%	1.011%	1.060%

Table 5. F3 FPGA Block RAM, Digital Signal Processing, Flip-Flop, and LookUp Table resource usage for common network monitoring tasks.

Memory usage in flow-level loss detection: We now vary k in the loss detection query to understand its memory usage for different loss scenarios (Figure 7). To track 1024 lossy flows with 99.99% probability, we only need ~2MB memory, which is around 16.5% of Xilinx U250 BRAM [1]. Note that 1024 lossy flow in a $100\mu s$ epoch is already a worst-case scenario.

6 Discussion

Dataplane-speed Reaction. Faster reaction time reduces the impact of bad network events (e.g., LDoS attack in Table 1). Since F3 is fully on the dataplane (switch+FPGA), we envision that F3 enables the opportunity for reaction in dataplane speed (e.g., 10 μs latency). In F3 FPGA, there could be one *reaction decision* module that decides the reaction strategy (e.g., rate limiting) based on analysis results and sends packets to inform the F3 switch. The F3 switch can perform a reaction based on the reaction strategy.

Commodity switch+FPGA. In our current F3 prototype, we use the programmable switch to implement the reduction operator prefix (ROP). We believe that F3 can also apply to commodity switch+FPGA architecture if switch vendors can provide programming APIs for operators to configure ROP.

F3 switch under heavy load. Our current F3 prototype relies on the programmable switch's traffic manager to generate and recirculate packets for reading counters and sending the data to the FPGA. However, when the switch is under heavy load, recirculation packets may be dropped, leading to the loss of monitoring information. This can affect the accuracy of F3's analysis on the FPGA side. Exploring methods to maintain the accuracy of F3's FPGA analysis under these conditions is an interesting direction for future work.

7 Related Work

In this section, we review the state-of-the-art approaches related to our work. We first discuss the ones relying only on programmable switches and then focus our attention on the solutions that leverage end-host systems. Finally, we briefly discuss proposals that use the combination of FPGAs and switches to accelerate network functions.

Switch-based telemetry. Switch-based telemetry systems leverage programmable switches to collect fine-grained telemetry data. Based on whether the analysis of the telemetry data needs the help of a CPU (switch CPU or end-host CPU), these telemetry systems can be classified into two categories. (1) Telemetry systems such as Sonata [30], Marple [59], FlowRadar [49], *Flow [71] and NetSeer [93] put part of the analysis logic of telemetry data on CPU. (2) Telemetry systems such as ConQuest [19] and ACC-Turbo [12] perform the analysis fully on the switch dataplane. The recent

Direct telemetry Access proposal [46] also shows that by doing switch processing and aggregation one can reduce the collection overheads for answering queries.

End-host based telemetry. End-host-based telemetry systems use end-hosts to collect and monitor telemetry data. Trumpet [58], PathDump [72], PingMesh [28], Confluo [40] leverage the advantage that end-hosts have ample resource and better programmability support. However, these systems lack the network visibility that may be necessary to debug a class of network problems.

Switch+FPGA Offloading. Tiara [83] is a stateful layer-4 load balancer that leverages the large high-bandwidth memory (HBM) available in state-of-the-art FPGAs and co-locates them with switches to offload memory-intensive tasks from the switch. Ribosome [64], instead, employs FPGAs to extend the packet-processing capabilities of switches and demonstrate its capabilities using packet-processing programs such as firewall, load balancer, and packet scheduler.

8 Conclusion

In this paper, we have presented F3, a new real-time telemetry solution relying on the cooperation between a switch ASIC pipeline and an FPGA co-processor colocated next to it. When designing F3, we carefully considered the gap between the two in terms of throughput and computation expressibility. To that end, we propose to split the query execution between the devices in a way that allows the switch program to remain as much as possible static while we re-program the FPGA when new queries are introduced. Further, we presented various techniques for bridging the throughput gap between the devices while minimizing the impact on the query accuracy. Finally, a testbed evaluation demonstrates that F3 misses up to 5.0× fewer DDoS attack vectors than ACC-Turbo, the state-of-the-art on-switch solution, and up to 24% fewer microburst-contributing flows for the same precision rate.

Acknowledgements

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. This work was also partially supported by the European Union - Next Generation EU under the Italian National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.3, CUP D43C22003080001, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART"). This work was also partially supported by NSF NeTS 2211383 and by a gift funding from Huawei.

References

[1] Alveo u250 data center accelerator card. https://www.xilinx.com/products/boards-and-kits/alveo/u250.html#specifications. Accessed: 2022-11-07.

[2] Cross-correlation wikipedia. https://en.wikipedia.org/wiki/Cross-correlation. Accessed: 2022-11-07.

[3] Intel tofino expandable architecture. https://www.intel.com/content/www/us/en/architecture-and-technology/intelligent-fabric/tofino-expandable-architecture-white-paper.html. Accessed: 2022-11-07.

[4] Intel® tofino™ programmable ethernet switch asic. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html. Accessed: 2022-11-07.

[5] ns-3 simulation. https://www.nsnam.org/. Accessed: 2022-05-09.

[6] Sequential k-means clustering. https://www.cs.princeton.edu/courses/archive/fall08/cos436/Duda/C/sk_means.htm. Accessed: 2022-11-07.

[7] Solving the mystery of link imbalance: A metastable failure state at scale. https://engineering.fb.com/2014/11/14/production-engineering/solving-the-mystery-of-link-imbalance-a-metastable-failure-state-at-scale/. Accessed: 2022-11-07.

[8] Tofino™ x: Intel® tofino™ intelligent fabric processor with intel® fpga. https://opennetworking.org/wp-content/uploads/2022/01/12-Server-Load-Balancer-Accelerator-P4_workshop_2021_Petr.pdf. Accessed: 2022-05-09.

[9] Virtex hbm fpga. https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus-hbm.html. Accessed: 2022-11-07.

[10] Vitis high-level synthesis. https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS. Accessed: 2022-05-09.

[11] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, page 19, USA, 2010. USENIX Association.

[12] Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. Aggregate-based congestion control for pulse-wave ddos defense. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 693–706, 2022.

[13] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 503–514, 2014.

[14] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74, 2010.

[15] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, Shir Landau Feibish, Danny Raz, and Minlan Yu. Routing oblivious measurement analytics. In *2020 IFIP Networking Conference (Networking)*, pages 449–457. IEEE, 2020.

[16] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. Designing heavy-hitter detection algorithms for programmable switches. *IEEE/ACM Transactions on Networking*, 28(3):1172–1185, 2020.

[17] Ran Ben Basat, Gil Einziger, Isaac Keslassy, Ariel Orda, Shay Vargaftik, and Erez Waisbard. Memento: Making sliding windows efficient for heavy hitters. *IEEE/ACM Transactions on Networking*, 30(4):1440–1453, 2022.

[18] Dehao Chen, David Xinliang Li, and Tipp Moseley. Autofdo: Automatic feedback-directed optimization for warehouse-scale applications. In *Proceedings of the 2016 International Symposium on Code Generation and Optimization*, pages 12–23, 2016.

[19] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. Fine-grained queue measurement in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 15–29, 2019.

[20] Yu Chen and Kai Hwang. Collaborative detection and filtering of shrew ddos attacks using spectral analysis. *Journal of Parallel and Distributed Computing*, 66(9):1137–1151, 2006.

[21] Ming Cheng, Ziyi Zhou, Bowen Zhang, Ziyu Wang, Jiaqi Gan, Ziang Ren, Weiqi Feng, Yi Lyu, Hefan Zhang, and Xingjian Diao. Efflex: Efficient and flexible pipeline for spatio-temporal trajectory graph modeling and representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2546–2555, 2024.

[22] Paul Emmerich, Daniel Raumer, Sebastian Gallenmüller, Florian Wohlfart, and Georg Carle. Throughput and latency of virtual switching with open vswitch: A quantitative analysis. *Journal of Network and Systems Management*, 26:314–338, 2018.

[23] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking:{SmartNICs} in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[24] Sumit Ganguly. Counting distinct items over update streams. *Theoretical Computer Science*, 378(3):211–222, 2007.

[25] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. A microscopic view of bursts, buffer contention, and loss in data centers. In *Proceedings of the 22nd ACM Internet Measurement*

Conference, pages 567–580, 2022.

[26] Soudeh Ghorbani, Zibin Yang, P Brighten Godfrey, Yashar Ganjali, and Amin Firoozshahian. Drill: Micro load balancing for low-latency data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 225–238, 2017.

[27] ETH Zurich Networked Systems Group. ACC-Turbo: Simulations. https://github.com/nsg-ethz/ACC-Turbo/tree/main/simulations. Accessed: 2024-09-27.

[28] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 139–152, 2015.

[29] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 357–371, New York, NY, USA, 2018. Association for Computing Machinery.

[30] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[31] Wenchen Han, Vic Feng, Gregory Schwartzman, Michael Mitzenmacher, Minlan Yu, and Ran Ben-Basat. Francis: Fast reaction algorithms for network coordination in switches. *arXiv preprint arXiv:2204.14138*, 2022.

[32] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 29–42, New York, NY, USA, 2017. Association for Computing Machinery.

[33] Brandon Heller, Srinivasan Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: Saving energy in data center networks. In *Nsdi*, volume 10, pages 249–264, 2010.

[34] Huawei. Telemetry. https://support.huawei.com/enterprise/en/doc/EDOC1100196389, 2021.

[35] Intel. Intel deep insight network analytics software. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/network-analytics/deep-insight.html, 2021. Accessed: 2021-06-10.

[36] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 2017.

[37] Jaeyeon Jung, Vern Paxson, Arthur W Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 211–225. IEEE, 2004.

[38] Pravein Govindan Kannan, Nishant Budhdev, Raj Joshi, and Mun Choon Chan. Debugging transient faults in data centers using synchronized network-wide packet histories. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 253–268, 2021.

[39] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. HULA: Scalable Load Balancing Using Programmable Data Planes. In *Symposium on Software Defined Networking Research (SOSR)*. ACM, 2016.

[40] Anurag Khandelwal, Rachit Agarwal, and Ion Stoica. Confluo: Distributed monitoring and diagnosis stack for high-speed networks. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 421–436, 2019.

[41] K Krishna and M Narasimha Murty. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3):433–439, 1999.

[42] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247, 2003.

[43] Jan Kučera, Ran Ben Basat, Mário Kuka, Gianni Antichi, Minlan Yu, and Michael Mitzenmacher. Detecting routing loops in the data plane. In *Proceedings of the 16th International Conference on emerging Networking Experiments and Technologies*, pages 466–473, 2020.

[44] Jan Kučera, Diana Andreea Popescu, Han Wang, Andrew Moore, Jan Kořenek, and Gianni Antichi. Enabling event-triggered data plane monitoring. In *Proceedings of the Symposium on SDN Research*, pages 14–26, 2020.

[45] Aleksandar Kuzmanovic and Edward W Knightly. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, 2003.

[46] Jonatan Langlet, Ran Ben Basat, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. Direct telemetry access. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 832–849, 2023.

[47] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 137–152, 2017.

[48] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 1–14, 2016.

[49] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 311–324, 2016.

[50] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Lossradar: Fast detection of lost packets in data center networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 481–495, 2016.

[51] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 44–58. 2019.

[52] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. SIGCOMM '16, page 101–114, New York, NY, USA, 2016. Association for Computing Machinery.

[53] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[54] Gonçalo Matos, Salvatore Signorello, and Fernando MV Ramos. Generic change detection (almost entirely) in the dataplane. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, pages 113–120, 2021.

[55] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2017.

[56] Tal Mizrahi, Vitaly Vovnoboy, Moti Nisim, Gidi Navon, and Amos Soffer. Network telemetry solutions for data center and enterprise networks. *Marvell, White Paper*, 2018.

[57] Edgar Costa Molero, Stefano Vissicchio, and Laurent Vanbever. Hardware-accelerated network control planes. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, HotNets '18, page 120–126, New York, NY, USA, 2018. Association for Computing Machinery.

[58] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 129–143, 2016.

[59] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 85–98, 2017.

[60] Maksim Panchenko, Rafael Auler, Bill Nell, and Guilherme Ottoni. Bolt: a practical binary optimizer for data centers and beyond. In *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14. IEEE, 2019.

[61] Sudarsanan Rajasekaran, Manya Ghobadi, Gautam Kumar, and Aditya Akella. Congestion control in machine learning clusters. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, pages 235–242, 2022.

[62] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, and Alex C Snoeren. Passive realtime datacenter fault detection and localization. In *NSDI*, pages 595–612, 2017.

[63] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, et al. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 735–749, 2020.

[64] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostic, and Marco Chiesa. A high-speed stateful packet processing approach for tbps programmable switches. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2023.

[65] Robert Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 207–212, 2004.

[66] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In *2019 International Carnahan Conference on Security Technology (ICCST)*, pages 1–8. IEEE, 2019.

[67] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, Anand Kanagala, Jeff Provost, Jason Simmons, Eiichi Tanda, Jim Wanderer, Urs Hölzle, Stephen Stuart, and Amin Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, page

183–197. Association for Computing Machinery, 2015.

[68] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. Re-architecting traffic analysis with neural network interface cards. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 513–533, 2022.

[69] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. DC.P4: Programming the Forwarding Plane of a Data-center Switch. In *Symposium on Software Defined Networking Research (SOSR)*. ACM, 2015.

[70] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, Shan Muthukrishnan, and Jennifer Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176, 2017.

[71] John Sonchack, Oliver Michel, Adam J Aviv, Eric Keller, and Jonathan M Smith. Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 823–835, 2018.

[72] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying datacenter network debugging with {PathDump}. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 233–248, 2016.

[73] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Distributed network monitoring and debugging with {SwitchPointer}. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 453–456, 2018.

[74] Mikkel Thorup. High speed hashing for integers and strings. *arXiv preprint arXiv:1504.06804*, 2015.

[75] Nguyen Van Tu, Jonghwan Hyun, Ga Yeon Kim, Jae-Hyoung Yoo, and James Won-Ki Hong. Intcollector: A high-performance collector for in-band network telemetry. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 10–18. IEEE, 2018.

[76] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, and T. S. Eugene Ng. Closed-loop network performance monitoring and diagnosis with SpiderMon. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 267–285, Renton, WA, April 2022. USENIX Association.

[77] Zeke Wang, Hongjing Huang, Jie Zhang, and Gustavo Alonso. Shuhai: Benchmarking high bandwidth memory on fpgas. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 111–119. IEEE, 2020.

[78] Henry S Warren. *Hacker's delight*. Pearson Education, 2013.

[79] Jonathan Woodruff, Robert N.M. Watson, David Chisnall, Simon W. Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G. Neumann, Robert Norton, and Michael Roe. The cheri capability model: Revisiting risc in an age of risk. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ISCA '14, page 457–468. IEEE Press, 2014.

[80] Liangcheng Yu, John Sonchack, and Vincent Liu. Mantis: Reactive programmable switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 296–309, 2020.

[81] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[82] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. Quantitative network monitoring with netqre. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 99–112, 2017.

[83] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, et al. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1345–1358, 2022.

[84] Junxue Zhang, Wei Bai, and Kai Chen. Enabling ecn for datacenter networks with rtt variations. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 233–245, 2019.

[85] Lu Zhang, Weiqi Feng, Chao Li, Xiaofeng Hou, Pengyu Wang, Jing Wang, and Minyi Guo. Tapping into nfv environment for opportunistic serverless edge function deployment. *IEEE Transactions on Computers*, 71(10):2698–2704, 2021.

[86] Lu Zhang, Chao Li, Xinkai Wang, Weiqi Feng, Zheng Yu, Quan Chen, Jingwen Leng, Minyi Guo, Pu Yang, and Shang Yue. First: Exploiting the multi-dimensional attributes of functions for power-aware serverless computing. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 864–874. IEEE, 2023.

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

[88] Teng Zhang, Jianying Wang, Xuntao Cheng, Hao Xu, Nanlong Yu, Gui Huang, Tieying Zhang, Dengcheng He, Feifei Li, Wei Cao, et al. {FPGA-Accelerated} compactions for {LSM-based}{Key-Value} store. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 225–237, 2020.

[89] Yinda Zhang, Liangcheng Yu, Gianni Antichi, Ran Ben-Basat, and Vincent Liu. Enabling silent telemetry data transmission with invisiflow. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025.

[90] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, and Nicholas Zhang. LightGuardian: A Full-Visibility, lightweight, in-band telemetry system using sketchlets. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 991–1010. USENIX Association, April 2021.

[91] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C Hoe, Vyas Sekar, and Justine Sherry. Achieving 100gbps intrusion prevention on a single server. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1083–1100, 2020.

[92] Wu Zhijun, Li Wenjing, Liu Liang, and Yue Meng. Low-rate dos attacks, detection, defense, and challenges: a survey. *IEEE access*, 8:43920–43943, 2020.

[93] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. Flow event telemetry on programmable data plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 76–89, 2020.

Received June 2024; revised September 2024; accepted October 2024