Traffic Monitoring Using Programmable Switch Hardware for In-Network Aggregation

Chris Misa
Area Exam
Advisors: Ramakrishnan Durairajan, Reza Rejaie

ABSTRACT

The ability to observe and make sense of network traffic is a persistent and well-established requirement for effective administration of performant and secure networks. Over the last ten years, the capabilities as well as the needs and requirements of monitoring network traffic have undergone considerable evolution sparking numerous research efforts. Three historically distinct background areas fundamentally shape this evolution: network monitoring system design, applications of programmable switch hardware, and in-network aggregation methods. An emergent body of research leverages aspects from all three of these background areas to develop systems that collect more detailed, precise traffic insights while simultaneously realizing order of magnitude improvements in traffic processing efficiency.

Given the increasing activity and potential for high impact, we focus on this emergent direction of research in *traffic monitoring using programmable switch hardware for in-network aggregation.* The three background areas whose intersection defines this research direction provide natural and intuitive dimensions along which each new work can be classified. We leverage these dimensions to construct a hierarchical taxonomy which groups efforts by types of network monitoring tasks considered, specifications and shapes of aggregation computations involved, and types of processing platforms targeted. For each sub-group implied by our taxonomy, we discuss common approaches, key contributions, and open problems apparent across works in that sub-group. Finally we reflect on open problems more generally across all groups proposing three concrete directions for future research.

1 INTRODUCTION

Effective management of modern computer networks requires observing and gaining insights into the traffic flowing through these networks. For example, network administrators need to know how much and what types of traffic their networks forward for design and planning purposes. Traffic experiencing reduced performance (e.g., high latency, packet loss) could be indicative of faulty network equipment or targeted denial of service attacks. These along with many other tasks motivate careful design and implementation of network traffic monitoring systems.

The last ten years have seen significant evolution in both the needs of network monitoring tasks as well as the capabilities and techniques available to address these needs. In particular, three background areas of research have a fundamental impact on recent network traffic monitoring proposals: *(i)* the types and definitions of traffic monitoring tasks considered; *(ii)* the techniques and design patterns for in-network aggregation; and *(iii)* the availability of programmable switch hardware as a processing platform for network monitoring.

First, the requirements of network monitoring tasks have evolved into complex computations over packet streams requiring filtering, aggregation, correlation, and other order-dependent operations. Instead of simply counting the number of packets transmitted on various links in the network (e.g., SNMP [43, 113]), modern network administrators need tailored solutions that provide fine-grained insights into their traffic to combat challenges like performance degradation caused by link contention [131, 140] or malicious attack traffic [39, 92]. These new requirements for traffic monitoring pose critical challenges in how to execute complex aggregation computations over the massive volume of data flowing through modern networks.

Second, programmable switch hardware [6, 28] emerged as a revolutionary tool for handling a wide variety of high data rate network processing problems. Programmable switch hardware exposes a fixed set of hardware primitive operators in comparatively high-level languages (e.g., P4 [27], NPL [9]) which can perform a wide variety of per-packet processing at extremely high throughputs (e.g., Tbps). However, leveraging programmable switch hardware for network monitoring is nontrivial due to the limited types of computations and state available, thus motivating research in adapting and inventing traffic monitoring algorithms to fit these constraints.

Finally, the process of using programmable switch hardware for network traffic monitoring is most effective when considered as in-network aggregation. The key idea is that since programmable switch hardware has the capability to update persistent state on a per-packet basis, its most effective use is as an in-network aggregator—this way, only aggregate results need to be exported to network administrators significantly increasing the efficiency of traffic monitoring systems. To realize this goal, works in this area grapple

with questions of what types of aggregation operations can be offloaded in this way and how to realize this offloading efficiently in end-to-end systems.

Viewing recent research efforts as an intersection of the three background areas described above clearly exposes the exciting promises driving research in this area of *traffic monitoring using programmable switch hardware for in-network aggregation*—when leveraged effectively, work in this particular area has the potential to revolutionize how network traffic is monitored and more generally how networks are run. Considering these three background areas also translates to three high-level *dimensions* along which each work in this area can be understood: the fundamental impetus of traffic monitoring translates to a dimension of different types of traffic monitoring tasks and different end consumers (e.g., traffic profiling vs. network automation); the in-network aggregation approach translates to a dimension of different forms of aggregation computation (e.g., different aggregation granularities, different summarizing functions per aggregate); the use of programmable switch hardware translates to a dimension of different choices of processing platforms surrounding the switch hardware ASIC (e.g., ASIC-only vs. ASIC and CPU-based post-processing).

Through careful observation of how works in the research literature populate and cluster in this three-dimensional space, we develop a detailed hierarchical taxonomy. The ordering, particular unbalanced structure, and temporal evolution of this taxonomy exposes several key insights into work in this area: we connect particular instances of unbalance to particular trends, assumptions, and constraints faced. We also connect the historical trends observed to particular developments in hardware technology (e.g., availability of Tofino-based [6] switch ASICs) and observe in particular that applications of traffic monitoring in network automation systems is the most recently initiated region of this space. We take this as an indication of the potential for network automation tasks to foster impactful future research efforts in this area.

After detailing the three background research areas whose three-way intersection we focus on in particular (§ 2) and presenting our taxonomy and related observations (§ 3), we perform a detailed analysis of each particular research direction implied by the leaves of our taxonomy tree (§ 4-6). For each direction, we present a list of papers constituting progress in that direction over the past ten years, summarise common design patterns and challenges considered, key contributions, and open problems. Finally we summarise key open problems across all directions and describe three concrete possible approaches for future research (§ 7).

2 DEFINITION OF THE AREA

As shown in Figure 1, this survey defines traffic monitoring using programmable switch hardware for in-network aggregation as the intersection of network traffic monitoring (**Area A**), in-network aggregation (**Area B**), and application of programmable switch hardware to networking problems (**Area C**). We next describe and give examples of each region in the figure to solidify connection to the larger research landscape implied by these background works.

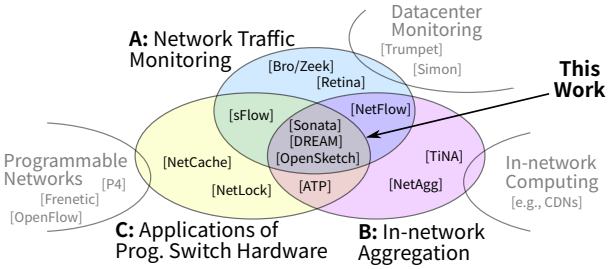

Figure 1: Diagram of how the target area fits into surrounding research areas.

Area A: Network Traffic Monitoring. Ideally, networks should be able to forward data with perfect accuracy and low, deterministic latency. However, in reality, this is not the case due to human inconsistencies in how the network is setup (e.g., misconfigurations), the possibility of component malfunctions (e.g., a packet is lost due to corruption during transmission), and unpredictable variation in communication patterns (e.g., natural variations in traffic, malicious attack traffic). Given these realities, nearly all networks must monitor that traffic they are forwarding using systems similar to the one shown in Figure 2 to determine when network performance may be impacted by any of the above causes. Due to large volumes of data continuously passing through the network, the primary challenge is in determining which particular aspects of traffic need to be monitored and how to efficiently execute this monitoring given a limited computation budget.

Traditional network monitoring systems like Bro [111] (now called Zeek [13]) receive a mirrored stream of individual network packets in software and perform monitoring computations using clusters of general-purpose CPU-based systems to perform the required computations. The key advantage of such systems is the flexibility to execute many types of analysis. Several domain-specific languages have been proposed to simplify specification of monitoring tasks (e.g., Chimera [26], Gigascope [38], NetQRE [149]) whereas other works push the limits of CPU-based processing (e.g., 007 [20], dShark [50], Confluo [80], Retina [137]). However,

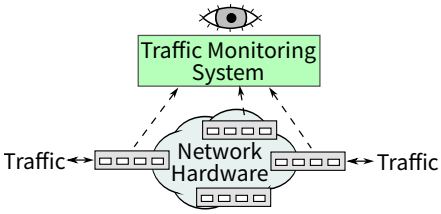

Figure 2: Network monitoring systems collect and report data about traffic flowing through networks.

the network bandwidth overheads of forwarding copies of all packets to a central server cluster, as well as the computing throughput required to process large volumes of traffic make this approach challenging and resource-intensive.

Area B: In-network Aggregation A second related category of work looks at leveraging processors (e.g., general-purpose CPUs) distributed around the network to perform intermediate data aggregation thus reducing the total volume of network traffic generated by an application. Example applications where this has been applied are wireless sensor networks (e.g., TiNA [122]), batch processing jobs in data centers (e.g., Camdoop [37], NetAgg [97]), and distributed machine learning (e.g., Parameter Hub [94]).

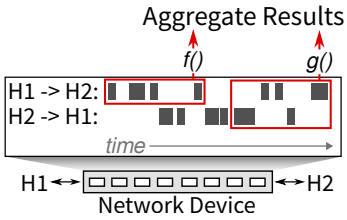

Figure 3: Example of two aggregation granularities: $f()$ computes per-source, destination pair whereas $g()$ computes over all traffic.

In the case of network traffic monitoring, aggregation typically refers to grouping packets based on common attributes (e.g., header fields) and computing summary metrics per-group (e.g., the total number of packets from the same source). Figure 3 shows a simplified example of this kind of aggregation. In this example, two hosts H1 and H2 send packets through a network device over time as shown in the box above the device. Two possible aggregation computations are shown: $f()$ will produce a summary metric for each distinct source, destination pair (e.g., H1 $\rightarrow$ H2) whereas $g()$ produces a single summary over a period of time for all traffic. Real in-network aggregation computations may be far more complex involving several different aggregation granularities, pipelining of aggregation stages, and/or reporting aggregate results per fixed time window or *epoch*.

Since in-network aggregation is inherently a summarising computation, some information about packets observed is unavoidably lost. However, there are two critical reasons why in-network aggregation is still a critical tool for modern network traffic monitoring systems. First, given packet rates in modern enterprise, ISP, and data-center networks, it is infeasible to mirror all packets to a centralized collector. As a result, many packet-level network monitoring systems are also forced to loose information due to low sample rates (e.g., one in a thousand [87]). Second, many network monitoring tasks inherently require aggregate metrics and the overheads of computing such metrics in CPU-based systems over mirrored traffic can be significant leading to high capital and operational overheads (see for example the detailed economic analysis in Jaqen [92]).

Area C: applications of programmable switch hardware. Networked systems research has a long trajectory of increasing control and flexibility over basic network operations which can be understood at a high level as programming the network. Software-defined networking (SDN) (e.g., OpenFlow [98], Frenetic [53]) begin the process of increasing network programmability by carefully splitting network operations between software and hardware components. In particular, software components make decisions about how to forward traffic while hardware executes simple forwarding rules which are dynamically controlled by software.

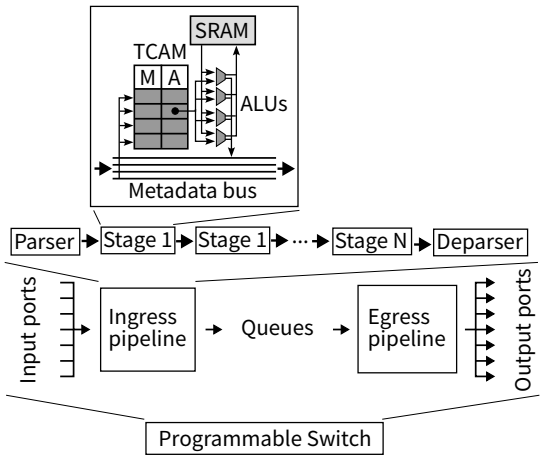

Figure 4: Simplified diagram of common programmable switch hardware processing model (e.g., RMT [28], Tofino [6]).

More recently, innovation in hardware design (e.g., RMT [28], Tofino [6]) opened up the possibility to perform far more

complex processing directly in the switch hardware ASIC. As shown in Figure 4, these switches typically forward packets between input and output ports using pipelines separated by a queuing stage. Each pipeline is constructed by a series of stages where each stage performs a single match-action operation using ternary content-addressable memory (TCAM) to match against packet metadata (e.g., headers) and a set of configurable arithmetic logic units (ALUs) which read and write from limited per-stage SRAM as well as the per-packet metadata bus. Switch programs typically consist of two parts: *(i)* a statically-compiled specification of which metadata fields are matched in each stage and of how ALUs perform actions and *(ii)* a dynamically installed list of match-action rules that select which pre-compiled action to execute when a packet matches in TCAM. Note that the action definitions can only use each ALU once and a hence limited to a constant number of primitive operations. Note also that stages (typically) do not share SRAM and hence information (e.g., intermediate results) must be passed in one direction between stages via the metadata bus.

Beyond research in hardware design and the design of appropriate hardware programming interfaces (e.g., P4 [27], Domino [124]), a significant body of work has looked at the question of how these new hardware capabilities can be leveraged for a wide range of tasks. For example, functions like NAT (e.g., Gallium [150], TEA [82]), load-balancing (e.g., Cheetah [21]), caches (e.g., IncBricks [89], NetCache [72]), distributed consistency (e.g., NetLock [147], NetChain [72], Paxos [41]) can be implemented entirely or partially in the switch ASIC leading to extreme gains in processing throughput and efficiency.

Area A ∩ Area B. In order to reduce the volume of traffic mirrored to software, several approachs perform initial aggregation of packets in processors closer to network forwarding such as routers and switch control CPUs. For example, the widely adopted NetFlow [35] standard first mirrors packets to a router CPU, then aggregates packets into flows (defined by common packet header field values) and exports statistics (e.g., packet counts) for each flow to a software collector. Although this reduces the volume of traffic exported, it still poses a significant computational bottleneck on the router or switch CPU. In most actual deployments network hardware only mirrors a sample of packets to the router CPU for aggregation again impacting accuracy of the resulting flow-level statistics.

Area A ∩ Area C. Another approach to reducing volume of mirrored traffic is to leverage switch hardware to only mirror a random sample of packets, and/or to only mirror important parts of packets like the headers (e.g., SFLOW [11]). Although this approach reduces the volume of traffic mirrored, it also degrades the accuracy of monitoring in ways that may render some monitoring tasks useless [96].

Area B ∩ Area C. Beyond network traffic monitoring, several works look at leveraging programmable switch hardware to perform in-network aggregation (e.g., SHArP [61], DAIET [118]), most recently for distributed ML training (e.g., SwitchML [119], ATP [84], Panama [56]).

Area A ∩ Area B ∩ Area C: traffic monitoring using programmable switch hardware for in-network aggregation. This work focuses on approaches at the three-way intersection of all three of these areas: using programmable switch hardware to perform some computation/aggregation directly in the network before exporting monitoring results. As demonstrated by several pivotal work in this are (e.g., Sonata [63]), this approach has the potential to both reduce computation overheads (since processing is performed directly in the packet-forwarding hardware) as well as the volume of exported traffic (since only aggregated or filtered monitoring results have to leave the network hardware data plane). The key challenges faced are how to map the computations required for traffic monitoring in to a limited set of hardware processing primitives (e.g., match-action tables) and how to allocate limited computational and memory budgets available in hardware to the wide range of monitoring tasks that must be executed (e.g., as dealt with in DREAM [103] or OpenSketch [146]).

2.1 Prior Surveys

Recent surveys [25, 75, 100] focus on programmable data planes and applications of the P4 programming language [59, 65, 78] summarizing current applied research areas where increased programmatic control over network processing has had notable impact. However, due to the wide variety of programming targets, interfaces, and applications considered, these surveys fail to provide precise characterization of challenges, solutions, and openings in the particular intersection considered in this work. In particular, their treatment of network monitoring problems mix programmable switch hardware systems with other programmable data plane targets (e.g., CPU-based virtual switches), hence obfuscating the particular challenges and complex relationships between hardware and software.

Another set of recent surveys summarises use of programmable data plane technologies in network security applications [17, 40, 55]. Again, these works freely mix the fundamentally different challenges of different data plane targets obscuring the particular challenges of programmable switch hardware. For example, only 38.6% of the works considered in [55] actually feature switch hardware methods. Several security-focused surveys also focus on the general security implications of systems built on programmable data planes [17, 42] which is an orthogonal and separate concern

to applications of programmable switch hardware to traffic monitoring.

To the best of our knowledge, ours is the first study to focus specifically on applications of programmable switch hardware as a specific subset of data plane programmability. By further limiting our study to aggregation-based network traffic monitoring tasks, we are able to perform deeper analysis of the key challenges and open problems faced by this particular research field.

3 OUR APPROACH

3.1 Taxonomy

We identify a set of 50 works from top-tier venues (e.g., USENIX NSDI [10], ACM SIGCOMM [12]) that represent the progression of research activity in traffic monitoring using programmable switch hardware for in-network aggregation. In order to clarify the structure of these works and to organize our discussion, we consider three orthogonal dimensions derived from the three surrounding areas: *(i)* the reason for why network traffic needs to be monitored (roughly corresponding to Area A); *(ii)* the type of aggregation computation performed (roughly corresponding to Area B); and *(iii)* the type of processors used (roughly corresponding to Area C). Figure 5 shows these dimensions as shaded columns and describes the points in each dimension as rows, additionally imposing a hierarchical classification structure described as a tree rooted on the left of the figure. The branches of this tree correspond to discrete groups of works which are discussed in the sections referenced and represented by the canonical examples on the far right.

The particular structure of Figure 5 is motivated by *(i)* an ordering of the dimensions from high-level decisions about what aspects of traffic should be monitored to low-level decisions about the particular processors to employ and *(ii)* observations of natural grouping of the considered works in each dimension.

Why Monitor Traffic? We identify three high-level reasons for monitoring network traffic—traffic profiling, network performance, and network automation—and further refine each reason based on common concrete applications.

First, *traffic profiling* (§ 4) seeks to collect particular metrics or features of traffic such as the total number of connections in a given time window or a list of connections sending above a certain rate (i.e., heavy hitters). These metrics are used in traffic profiling to understand normal network behaviors, to drive network architecture and planning decisions, and to detect anomalous traffic patterns associated with network-based attacks. Some traffic profiling works collect a single metric (§ 4.1) where as others are able to collect a large number of metrics (§ 4.2) in a single unified system.

Second, *network performance* (§ 5) seeks to monitor and report when network events such as congestion, device failures, or routing imbalance impact forwarding performance in well-defined network settings. These systems are typically deployed in data center networks where their outputs are used to quickly identify and respond to networking problems such as configuration bugs or device failures as well as to localize performance issues. Some systems in this category focus only on detection of performance events (§ 5.1) whereas others additionally seek to provide auxiliary debugging information (§ 5.2).

Finally, *network automation* systems (§ 6) monitor network traffic in order to react to particular traffic events by automatically changing the network's processing behavior. In these systems, traffic monitoring is closely motivated by and tied to the particular goals which the automation system seeks to ensure and the particular types of actions available to the automation system. We consider two examples of such systems: DDoS defense (§ 6.1)—which monitors traffic to detect, isolate, and mitigate DDoS attack traffic—and flow offloading (§ 6.2)—which selects traffic subsets to offload to processors with different performance characteristics.

Type of Aggregation? All in-network aggregation works define particular specification of the aggregation computation including specification of how aggregates are defined, how packets in each aggregate are summarised, and when per-aggregate results are collected.

In the simplest case of single-metric traffic profiling, the per-aggregate computation is already defined by the particular metric of interest, hence the primary distinction at this level is whether the system assumes a single, fixed aggregation granularity, e.g., five-tuple (§ 4.1.1) or if the system supports multiple, simultaneous aggregation granularities, e.g., different prefix lengths for different subsets of traffic (§ 4.1.2).

In the more complex case of multi-metric systems, we identify four high-level patterns. Some works leverage inherent commonalities between different metrics (e.g., entropy and cardinality) to compute multiple metrics from a single common summary data structure via post-processing (§ 4.2.1). Another type of work offers a pre-defined menu of fixed aggregation computations (including both aggregation granularity as well as other parameters like aggregation function/metric, duration between result reports, etc.) from which network administrators may choose a subset for deployment (§ 4.2.2). Other works allow network administrators to specify more complex queries in a domain-specific language (DSL) and compile language primitives into switch hardware and other processing platforms, e.g., CPUs for post-processing (§ 4.2.3). Finally, a recent set of works enable queries (either chosen from a fixed menu or specified

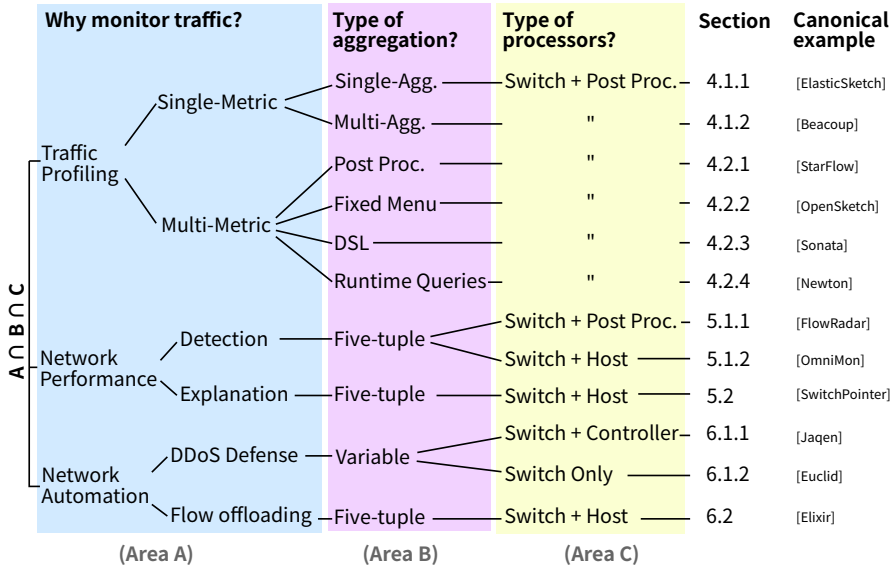

Figure 5: Hierarchic taxonomy proposed to group works dealing with the use of programmable switch hardware to perform in-network aggregation for traffic monitoring at the intersection of areas A, B, anc C.

in a DSL) to be dynamically added and removed during runtime by installing more flexible "interpreters" into switch hardware (§ 4.2.4).

Type of Processors? Although all works considered in this paper use programmable switch hardware, most works also incorporate additional processing capabilities which further shapes the particular problems considered in each particular work.

Which additional processors are used is often directly determined by the particular reasons for monitoring. In traffic profiling works (§ 4) the switch performs initial aggregation stage(s), but additional processing is often required before the exact profiling results can be obtained. Network performance monitoring systems typically target datacenter and/or cloud networks where both network infrastructure as well as end-host infrastructure are controlled by the same administrative domain, hence these works can either perform all monitoring and aggregation in switch hardware with additional post processing as for profiling works (§ 5.1.1) or they can choose to offload certain parts of monitoring and aggregation to the end-host network stack (§ 5.1.2). Finally, in network automation works, the tight binding between traffic monitoring and updates to network forwarding policy necessitate particular placement of traffic processing. Since DDoS attacks break network connectivity by flooding in-network processing elements, traffic monitoring must be performed entirely in switch hardware. In some works, a CPU-based centralized controller periodically updates switch hardware processing to respond to attack scenarios (§ 6.1.1) whereas in others switch hardware enacts mitigation independently (§ 6.1.2). Flow offloading on the other hand processes some flows in hardware and others in software and hence must monitor network traffic both at the hardware and software vantage points to intelligently identify an optimal set of flows to process in switch hardware (§ 6.2).

Observations. The unbalanced nature of the tree structure shown in Figure 5 exposes several key trends in recent research at the intersection considered. First, in order to compute a wider variety of diverse metrics, traffic profiling works tend to rely on a common approach of collecting concise traffic summaries in switch hardware and leveraging more flexible CPU-based post-processing. This trend is reflected by the relative density under the traffic profiling branch in the "type of aggregation" dimension combined with the relative uniformity of this branch in the "type of processors" dimension. Second, network performance and network automation tasks tend to imply fixed notions of aggregation, but lend themselves to more creative exploration in terms of which types of processors are leveraged. This trend is reflected in the uniformity of these two branches in the "type of aggregation" dimension and relative density in the "type of processors" dimension. Finally, we note that flow offloading is a relatively recent addition to the switch hardware network monitoring literature and hence has not yet developed as rich of a structure in the three dimensions considered here compared to other directions.

Year:	2013	’14	’15	’16	’17	’18	’19	’20	’21	’22
Key events:	RMT [28], Tofino [6]	P4 [27]								
§ 4.1.1					[125]	[32, 64]	[71, 83]	[23, 121, 133]		
§ 4.1.2								[33]	[152]	
§ 4.2.1				[91]		[67, 126, 127, 145]		[128]	[68]	
§ 4.2.2	[146]									[14, 106]
§ 4.2.3					[108]	[63, 112]		[86]	[24]	
§ 4.2.4		[103]	[104]					[157]		[102, 154]
§ 5.1.1				[87, 88]		[74]		[134, 155, 156]		
§ 5.1.2								[69]	[153]	[54]
§ 5.2			[130]	[131]		[132]				[140]
§ 6.1.1								[151]	[92, 144]	
§ 6.1.2								[39]		[16]
§ 6.2									[110]	[141]

Table 1: Publication years of the 50 works considered in the intersection described by Figure 1 partitioned according to the taxonomy describe by Figure 5.

3.2 Historical Timeline

Although research in using SDN-based systems (e.g., OpenFlow [98]) has a long history stretching back till at least the 2000s (see [19] or [49] for in-depth treatments of the origins of SDN), the ability of switch ASICs to perform a custom set of in-network aggregations is a relatively recent development. Starting in 2013, several critical developments led to a flourish of innovation in this area including *(i)* seminal work on reconfigurable match action tables [28], a hardware model that combines the flexibility of FPGAs [8, 107] with the forwarding efficiency of Ethernet switch ASICs, *(ii)* novel programming interfaces like P4 [27], and *(iii)* a startup company with a strong commitment to supporting the research community (Barefoot Networks [1][1]). Table 1 traces the timeline of research during this period in traffic monitoring using programmable switch hardware for in-network aggregation, also showing the division of works considered among the leaves of Figure 5. Figure 6 summarises this data by showing the total number of publications per year.

Observations. Several high-level trends are apparent from this table.

First, traffic profiling works, in particular OpenSketch [146] and DREAM [103], had already developed initial approaches to in-network aggregation for traffic monitoring using previous-generation technology (FPGAs and TCAM-based counters on SDN switches respectively). These works mark the tail-end of classic SDN work which was typically limited to counter-based methods (e.g., [48, 73, 79, 148]) or required custom-designed hardware as in OpenSketch.

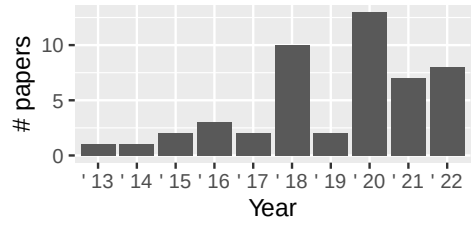

Figure 6: Summary of data from Table 1 showing the number of papers published each year.

Next, we note that the P4 programming interface [27] seems to have been a key enabler of the development of switch hardware applications research in the years after introduction of RMT/Tofino. In particular, starting in 2016 nearly all works use P4 as the interface for specifying switch hardware programs. We note that several works from 2016 on (e.g., UnivMon [91], FlowRadar [87]) are limited to evaluation on the P4 behavioral simulator [2] and are in some sense “pre”-switch hardware methods.

Finally, we note that specific applications of programmable switch hardware in the two network automation applications considered here (§ 6.1 DDoS defense and § 6.2) only recently gained traction in the research community (since 2020). We associate this with a trend from more general proposals with less specific network settings (e.g., UnivMon [91], Sonata [63]) towards more specialized proposals targeteding specific network settings (e.g., BufScope [54], Elixir [141]).

[1]Barefoot Networks was acquired by Intel in 2019 which recently (as of winter 2023) halted development of Tofino programmable switch ASICs [5, 7].

Overall, from Figure 6 we note that the rate of publication in this particular are appears qualitatively to be increasing over the time period considered.

3.3 Per-Group Methodology

In the following sections (§ 4, § 5, § 6), we discuss each leaf of the tree shown in Figure 5. Our discussion in each section is organized at a high-level by the following method. We first describe the group by detailing its particular placement in the taxonomy and its overall features with respect to the area defined in § 2. We then describe common design patterns observed across the set of works constituting the group. Next, we describe key challenges and contributions made towards solving these challenges in individual or groups of works. In most cases we describe contributions in chronological order following the logical progress of research, however, in some cases we deviate from strict chronological order in order to illustrate commonalities between works. Finally, we discuss limitations of the group's works in terms of a concrete set of one ore more open problems.

4 TRAFFIC PROFILING

Although network behavior is driven by the individual packets traversing network devices and links, due to extremely high packet rates (e.g., tens of millions per second) it is often infeasible and uninstructive to derive insights about network traffic directly from a listing of packets traversing the network. As a result, traffic profiling works develop methods to efficiently aggregate and summarise groups of packets to produce high-level metrics reflective of the network's current behaviors. Such metrics are critical for a wide range of network administration tasks such as capacity planning, performance tuning, and security event detection. We consider here the recent subset of research addressing this network profiling question that leverages programmable switch hardware to perform at least some part of the required aggregation and summarizing computations directly in the network itself.

4.1 Single-Metric Systems

The simplest type of traffic profiling systems focus on computing a single well-defined metric. Typically such systems seek to maximize the accuracy achievable for a given switch hardware computational capability, or conversely, to minimize the switch hardware resources required to achieve a particular accuracy goal.

4.1.1 **Single (Fixed) Aggregation Granularity.** Single-metric, single-aggregation granularity traffic profiling systems focus on how to adapt the computations required for a particular type of traffic profiling computation into the limited processing model and processing resources of programmable switch hardware.

Common design patterns. In order to perform aggregation using a fixed number of operations per packet and a fixed amount of high-speed memory accessible from packet processing logic, these works all implement different versions of lossy hash tables known colloquially as "sketches" [31, 36, 101]. Rather than dealing with hash collisions using linked-list buckets, probing, or other methods like Cuckoo hashing [109] which all require non-constant operations per update, these designs embrace hash collisions and find creative ways to deal with the consequences. As a result of this design pattern, these works all have some high-speed update method which is compiled into switch hardware to produce tables of counters for a fixed granularity and a fixed metric. These counters are periodically pulled to a CPU-based collector which performs additional statistical procedures on counter values to produce the final estimated traffic profile metrics.

A key challenge in this design space is the fact that sketch algorithms don't store the observed flow keys by default. In particular, the classic solution of maintaining a heap or other tree-like set data structure used in other applications of sketch algorithms (e.g., in databases [58, 66]) requires non-constant operations per update and is hence not possible in switch hardware. The works in this section develop novel solutions to this problem when addressing metrics that require flow keys such as heavy hitters, or define metrics overall all flow keys such as entropy.

Key contributions. Perhaps the first solution to compute a traffic profiling metric entirely in switch hardware, in particular heavy-hitters, is HashPipe [125]. HashPipe starts with a well-known algorithm to approximately computes the top-n elements in a data stream [99], then makes several modifications and relaxations of the original algorithm—in particular, sampling to estimate the minimum value and pipelining to fit the minimum estimation into a fixed number of switch hardware stages. Several subsequent works further improve heavy hitter detection on switch hardware to coordinate detection of network-wide heavy hitters by dynamically adjusting thresholds at each switch [64] and to use limited recirculation of packets [23] to fix the critical inefficiency in HashPipe of always evicting an entry (even for flows with a single packet).

Whereas HashPipe and the works mentioned above focus on packet or byte count heavy hitters, SpreadSketch [133] focuses on heaviness in terms of distinct entities associated with each flow (e.g., distinct sources contacted by each destination), adopting a similar approach of maintaining candidate flow keys associated with sketch counters. Snappy [32] also extends the notion of packet or byte count heavy hitters

to include subtraction in order to identify which flows make the largest contribution to long queues. However, Snappy does not maintain flow keys in switch memory, but acts directly on packets given the current estimate of the packet's flow's rate from the sketch counters (an approach also discussed in § 6.1.2).

Beyond variants of heavy-hitters, several other metrics have been computed in switch hardware. Following a similar methodology as HashPipe, QPipe [71] uses a near-optimal existing algorithm called SweepKLL [70] for computing quantiles. However, instead of modifying or relaxing SweepKLL, QPipe instead develops a novel technique of using unsampled packets as "workers" to facilitate the required arg-min computation between stages. A similar methodology is followed in [83] to implement a near-optimal entropy-estimation algorithm using a look-up approximation to the required maximally skewed alpha-stable distribution. Dart [121], on the other hand, develops a new approach using heuristics and lazy cache eviction to decide which SEQ packets in one direction of a TCP flow are most likely to produce valuable RTT estimates when paired with their corresponding ACK packets from the opposite direction.

Open problems. The first clear open problem is that of coverage: there are many more traffic profiling metrics (e.g., bandwidth estimation [34, 45, 52]) that could be useful even when computed in the limited single-aggregation form assumed in this section.

The second open problem is a less obvious consequence of the prevalent use of sketch-based techniques in these works. While sketch-based techniques do have rigorously-defined statistical error guarantees [31, 36], closer examination of these guarantees indicates that the accuracy of most sketches is a function of the total number of distinct keys tracked. In order to both decide on an appropriate memory allocation for particular sketches as well as to interpret the accuracy of retrieved sketch counters, network administrators first need to know the ground truth number of keys collected in the sketch. This leads to critical problems in real network settings where the number of distinct keys (e.g., the number of currently active flows) changes rapidly and is hard to measure directly. Sketch-based approaches must be made more robust against changes in the number of keys, or that other approximation methods must be considered.

4.1.2 **Multiple Aggregation Granularities.** In several cases, network administrators need to compute the same metrics over different sets of traffic and/or at different granularities. For example traffic could be aggregated both by source addresses and (independently) by destination addresses in the same system whereas single aggregation granularity systems would require one instance for aggregating source addresses and a separate, independent instance for aggregating destination addresses. Here we consider works that address this need head on by computing single, fixed traffic profiling metrics, but exposing interfaces to allow collecting these metrics over different traffic slices and aggregations.

Common design patterns. The works in this section define a limited universe of possibilities which can be understood as a global key space. Individual monitoring tasks or queries all compute the same metrics, but can be parameterized by different subsets as well as different aggregation granularities within this global key space. The key challenge faced in these works is how to design, implement, and analyze data structures that can collect profiling data for multiple aggregation granularities simultaneously with greater efficiency that simply capturing each granularity independently (e.g., with methods from § 4.1.1).

Key contributions. Beaucoup [33] makes perhaps the first contribution in this space by developing an approach to answer multiple threshold-based heavy-distinct-count queries where each query has a different aggregation granularity. For example, Beaucoup can simultaneously monitor for source addresses that contact more than a given number of destinations as well as destination addresses that contact more than a given number of sources. The key enabling idea is to interpret the heavy-distinct-count metric computation as a coupon-collector problem where each packet is associated with a hash-based "coupon" and each query, key pair is a coupon-collector. Given a fixed number of coupons and query, key pairs, Beaucoup then describes a simple mapping of the coupon collector problem into programmable switch hardware.

CocoSketch [152], on the other hand, addresses the problem of computing per-flow packet and byte counts for a flexible range of flow definitions (e.g., five-tuple-based flows, source-based flows, etc.). The key idea is to map the per-flow counting problem into the subset-sum estimation problem and apply the Unbiased SpaceSaving algorithm [135]. As with HashPipe [125], CocoSketch also applies modifications and relaxations to remove circular dependencies in the Unbiased SpaceSaving algorithms making it computable in switch hardware.

Open problems. Similar to fixed-granularity approaches discussed in § 4.1.1, a fundamental open problem in these works is how to apply variable granularity approaches to more types of metrics. The relative sparsity of this section is a testament to the fact that this is a significantly harder challenge compared to computing traffic profiling metrics at fixed granularity.

Also similar to the fixed-granularity approaches, the works in this section compile fixed-sized data structures (e.g., for a given fixed number of coupon-collectors in Beaucoup) into

programmable switch hardware while the underlying number of monitored keys inherently varies leading to resource allocation and accuracy estimation challenges. For works supporting multiple aggregation granularities, this problem is even more challenging to address because each monitored aggregation granularity may have different numbers of distinct aggregate groups (e.g., a different number of sources and destinations) so allocation must be made per aggregation granularity.

4.2 Multiple-Metric Systems

Given the fact that in many network scenarios, network administrators need to observe multiple traffic profile metrics, several works explore how multiple different metrics can be computed simultaneously in a unified system using one or more programmable switches. Following our taxonomy's leafs (Figure 5) we discuss these works according to different types of aggregation.

4.2.1 **Select Metrics From Post-Processesing of Fixed Dataplane Results.** Several works leverage the observation that in addition to per-flow packet and/or byte count estimates, sketch counters collected from switch hardware can also be used to estimate several higher-level metrics through CPU-based post processing. These works enable network administrators to compute a fixed set of a few traffic profiling metrics simultaneously through the same switch hardware computations.

Common design patterns. The works in this section develop single sketch-based data structures that are updated directly in switch hardware, then post process the resulting sketch counters on CPU-based systems to estimate a fixed set of metrics typically including heavy hitters, flow-size distribution summaries, entropy estimates, and cardinality estimates (e.g., the number of distinct flows). Note that all of these estimates are limited to the aggregation granularity at which the data plane sketch is compiled. For example, if the sketch is compiled per-destination address, then post processing can extract heavy destination addresses, entropy of estimations, or the number of distinct destinations, but cannot extract metrics w.r.t. other aggregations (e.g., w.r.t. source addresses).

Key contributions. UnivMon [91] heralded the idea of being able to produce multiple metrics through post-processing a single simple data structure through the idea of "universal sketching". Universal sketching builds on streaming algorithms research [29, 30] and essentially allows estimating a class of metrics that can be expressed as sums of a function of the frequencies of individual elements in the data stream (with the additional requirement that these functions must be upper-bounded by the second frequency moment). In practice, the set of metrics computable through this method corresponds to heavy-hitter-type metrics as well as global metrics like cardinality and entropy. Note that several further optimizations of the switch hardware implementation of UnivMon are discussed in [92, 93, 143].

Despite the novel theoretic underpinning, UnivMon has several limitations which were picked up in subsequent works. SketchLearn [67] develops a method to automatically extract flow keys from a multi-level sketch using statistical modeling techniques. ElasticSketch [145] explicitly deals with changes in the number of keys and distribution of traffic across keys commonly observed in network traffic by splitting the sketch data structure into heavy and light parts with different update policies. Most recently FCMSketch [128] and [68] develop methods for further improving the accuracy of similar sketch-based approaches using a hierarchical tree-like organization of sketch counters and a correspondence with compressive sensing respectively.

Another approach taken in StarFlow [127] and TurboFlow [126] takes the late-binding principle farther, using switch hardware only to group packet-level data into aggregates (e.g., flows), then exporting succinct summaries of all packets in each aggregate (e.g., StarFlow's "grouped packet vectors"). Although this approach enables a wider variety of metrics compared to sketch-based works, it also imposes higher load on the CPU-based post processing system and makes limited use of switch hardware's aggregation capabilities.

Open problems. As with the single, fixed aggregation solutions described in § 4.1.1, the works in this section all statically fix a particular aggregation granularity in the switch hardware program. In many ways this limits the universality of proposals like UnivMon [91] since separate independent instances must be maintained for all aggregation groups network administrators wish to profile. In a similar way, the use of sketches in may of these works is still plagued by the fundamental connection between sketch accuracy and the variable and unpredictable number of flows observed in network traffic. Importantly, although ElasticSketch [145] appears to address this issue head-on, their solution is still somewhat brittle (using a strict two-tier hierarchy) and focuses more on adapting to changes in traffic rate in software versions of the same algorithms.

4.2.2 **Select Metrics From Fixed Menu.** The works in this section allow network administrators to construct custom traffic monitoring switch hardware programs by selecting different types of metrics (e.g., heavy hitters, cardinality) from a fixed menu and parameterizing each selection to look at particular slices of traffic and at particular aggregation granularities.

Common design patterns. These works start by defining a set of metrics typically implemented through sketch-based algorithms then define methods for combining multiple algorithms in a single switch hardware program. The key

challenge faced in these approaches is how to minimize the switch hardware resources required to implement a given set of metrics or sketches. Although sketches on the surface seem to perform essentially the same types of computations and should be trivial to merge, when differing traffic slices, keys, and sketch algorithms are involved, creative combination strategies are required.

Key contributions. OpenSketch [146] demonstrated the first such system defining a flexible FPGA-based data plan that performs hashing, TCAM-based classification, and counting primitives and a library-based control plane containing preset formula for how to configure the data plan primitives for a variety of traffic profiling metrics. OpenSketch also proposed an active "sketch manager" which performs additional traffic profiling to tune memory allocations of the currently running profiling queries. For example, the sketch manager might run distinct-count profiling to automatically adjust the number of counters assigned to a heavy hitters query as the underlying number of keys changes.

Several recent works including HeteroSketch [14] and SketchLib [106] propose similar methods for coordinating execution of multiple sketch-based traffic profiling queries in a common system and for managing their resource allocations. In addition to targeting the modern generation of P4-programmable switches (instead of requiring an FPGA) and incorporating support for sketch algorithms adopted after OpenSketch (e.g., UnivMon [91]), these works also make several targeted resource usage improvements [106] and propose a unified framework for understanding the processing performance of sketch algorithms on diverse hardware and software targets [14].

Open problems. As with other traffic profiling works, there is a continuous open problem of incorporating support for more diverse metrics into the the "fixed menus" of works in this category. Also, although OpenSketch [146] included provisions for dynamically adjusting sketch counter allocation in response to changing numbers of flows, more recent works [14, 106] do not provide adequate solutions to the general challenges with provisions sketch counters and estimating accuracy of sketch-based results.

4.2.3 Define Metrics in Domain Specific Language.

Another approach to developing traffic profiling systems that collect multiple metrics is to allow network administrators to specify the metrics they seek to collect by combining a fixed set of primitive operations and combinators in a domain specific language. Expressions in this languages are called "queries" and a list of queries can be compiled for and installed into switch hardware to collect exactly the metrics required by network administrators.

Common design patterns. Rather than providing a fixed menu of metrics, the works in this section develop more flexible primitive operations, often specified in a stream-processing-like language, and allow network administrators to combine primitive operations to precisely specify the computations required for their metrics of interest.

The key challenges are in *(i)* how to define these languages to be expressive enough to allow useful metric computations but limited enough to still be compilable to the limited switch hardware processing model and *(ii)* given such a language and compiling method, how to optimize hardware resource usage in the face of unknown and unpredictable traffic loads.

Key contributions. Marple [108] initiated the DSL-based approach considered in this group by proposing "language-directed hardware design". In particular, Marple defines a stream-processing language with functional "map", "filter", "group-by", and "zip" primitives and describes how each of these primitives can be mapped into switch hardware programs (or in the case of "group-by" a combined hardware, software construct). Sonata [63] extends Marple by partitioning the language primitives of given queries between the switch hardware program and a CPU-based post-processor, thus breaking the strict limits on query complexity imposed by switch hardware limits in Marple. Concerto [86] and DynamiQ [24] further build on Sonata adding support for deploying queries across distributed programmable switches and enhanced supported for automatically adjusting resource allocations in response to changes in traffic composition respectively. EQuery [112] also proposes an expressive event-based language which can be compiled to efficient NFA, but does not appear to poses a concrete switch hardware implementation.

Open problems. Although the works in the section follow a programming-language-based approach to network traffic profiling, they connect only superficially to the rich literature on programming language theory. In particular, providing formal semantics as well as investigating other compiler-level optimization techniques remain open problems. Given the recent interest in formalizing low-level switch programming interfaces [18, 44, 81, 116], this may be a promising direction for future work.

Additionally, since aggregation operators are still implemented with sketch-based primitives (e.g., Sonata's [63] "reduce" operator), these works also face issues with provisioning and error estimation. While works like DynamiQ [24] make some progress in this direction, they still focus on optimizing processing performance (e.g., reducing the volume of tuples exported to the CPU-based collector) and leave result accuracy as a secondary concern. The flexibility provided by these language-based approaches further complicates sketch error estimation since reduction operations can be performed on arbitrarily filtered substreams and/or pipelined for multiple levels of aggregation.

4.2.4 **Support Runtime Changes.** Whereas works in the previous categories focus on statically configuring switch hardware to collect particular sets of metrics, another type of work seeks to enable network adminstrators to dynamically add and remove metrics or queries during runtime.

Common design patterns. In this approach, switch hardware acts more as an interpreter which is dynamically configured by a control plan to execute different aggregation computations over time. In particular, the programs compiled into switch hardware employ an extra level of indirection to allow switching between different computations based on the values of different ASIC "control" registers. A software controller can then control the per-packet computation without reinstalling the hardware program by simply writing different values to the control registers.

These works face similar challenges in balancing expressiveness of their query interfaces with feasibility of implementation through register-controlled computations. Additionally, whereas in previous approaches switch resources are statically allocated at compile time, works in this section face a dynamic resource allocation and/or scheduling challenge.

Key contributions. Initial efforts in DREAM [103] and SCREAM [104] leveraged fixed sets of TCAM-based counters in the previous generation of SDN-programmed switches to allow runtime control of multiple parallel instances of three types of queries (heavy hitters, hierarchical heavy hitters, and change detection) parameterized by traffic slice and aggregation granularity. In addition to supporting addition and removal of queries, these works actively adjust resource allocation among all running queries based on accuracy signals derived for the three particular query types.

More recently, works like Newton [157] and Flymon [154] developed methods for dynamically adding and removing arbitrary queries written in Sonata-like [63] stream processing languages from switch hardware during runtime. The key contribution of these works is the design of switch hardware programs that bake enough flexibility into the hardware pipeline so as to allow a network administrator to execute arbitrary stream processing queries simply by runtime configuration. DynATOS [102] further extends these works by developing a novel approximation and resource scheduling approach to execute the first aggregation stage in such runtime-controlled query systems while adapting to changes in query work load as well as changes in the underlying network traffic.

Open problems. A key open problem in this research direction is how to implement more generic approximation and resource management techniques that can adapt to complex query definitions with multiple levels of aggregation (note that the methods described in DynATOS [102] only apply to the first level of aggregation regardless of the complexity of the target query). This challenge also relates to the previously mentioned challenges with sketches: either methods to dynamically adjust sketch counter allocations must be developed, or entirely new approximation approaches must be explored (e.g., based on sampling theory as proposed in DynATOS [102]).

5 NETWORK PERFORMANCE

Proliferation of complex cloud-based distributed system architectures such as the micro-service model as well as performance-sensitive network applications such as video conferencing continue to raise the stakes for network performance. Cloud and other large-scale data center networks in particular face significant challenges in detecting and debugging performance events due to large numbers of end hosts, links, and switches. Detailed and accurate performance monitoring is critical in these settings in order to localize the cause of application-level performance issues (e.g., video freezes) to software performance or network performance.

We note that following the definitions set forth in § 2, several widely known approaches to network performance monitoring such as active approaches (e.g., PingMesh [62]), end-host-only approaches (e.g., Trumpet [105] and Simon [57]), and packet-mirroring approaches (e.g., EverFlow [158]) are considered out of scope in this paper and not considered in the following.

5.1 Detecting performance events

A large body of work develops systems to detect when performance events such as packet loss or increased latency occur and to identify the particular subsets of traffic (e.g., a particular set of flows) impacted. The results of this monitoring often take the form of alert-producing systems that log performance events for network administrators to use when debugging reported application-level performance issues. For example, if a client is experiencing issues with freezing video conferencing, the network administrator might check the logs of the network performance monitoring system to determine if lost packets of increased forwarding latency were the cause of the freeze or if a software-level issue is at fault. This section focuses in particular on the recent subset of performance monitoring systems that leverage programmable switch hardware for in-network aggregation.

5.1.1 **Switch Only.** The first group of performance monitoring systems relies entirely on switch hardware to observe network traffic and to perform initial aggregation computations, typically grouping packets into flows and reporting flow-level performance information to a collector for post-processing and event reporting.

Common design patterns. The systems in this section filter and aggregate particular performance-related events in switch hardware, then use CPU-based software (running on the switch's CPU and/or on a centralized collector) to correct for possible errors made by the lossy hardware approximations (e.g., hash collisions). The network-wide perspective along with inherent redundancy in the targeted data center networks allows disentangling such errors in way which are not possible in the single-switch scenarios considered in § 4.
Key contributions. FlowRadar [87] and LossRadar [88] make the primary contributions in this group leveraging Invertible Bloom filter Lookup Tables [60] and Invertible Bloom filters [47] respectively. These works capture per-fivetuple counters over very short time scales (e.g., 10 ms) using encoded flowsets collected in switch hardware and network-wide decoding. The fine aggregation as well as temporal granularity enables detecting a wide range of performance events (in the CPU-based post processing layer) including transient block holes, forwarding errors, ECMP load imbalance.

Rather than exporting flow-level metrics and performing event detection in post-processing, several more recent works seek to execute event detection logic directly in the data plane and only export records describing particular detected events to a central collector. BurstRadar [74] initiates this trend by detecting microbursts at port-level in the egress pipeline and emitting grouped burst snapshots summarizing all packets responsible for the burst. HyperSight [155] and NetSeer [156] extend this idea to several other performance events (including excessive forwarding delay, degraded throughput, long queues, out-of-order packets, and packet loss) by using a novel Bloom filter queue algorithms and circular-buffer aggregation respectively. PacketScope [134] offers even deeper visibility into performance events on a single switch by extending Sonata's [63] stream processing language with primitives connected to different pipelines and queuing stages within the switch ASIC.
Open problems. A key open problem with works in this category is how to connect performance events detected at the network level to higher-level performance metrics (e.g., at the application level). A somewhat related open problem is that although the works in this section target data center networks in particular, they do not provide comprehensive support for integrating with and interpreting the complex layered logical structure of data center traffic, for example virtual private clouds as discussed in [110].

5.1.2 **Switch Plus End Host.** The second group of performance monitoring systems observes network traffic both in switch hardware programs as well as in the network stacks of end hosts. This allows the system to augment the packet processing and aggregation capabilities of switch hardware (which have strict limits on memory and number of operations) with more flexible CPU and memory resources on end hosts.
Common design patterns. Works in this group coordinate per-flow, per-switch monitoring by using in-band methods initiated from and collected at end host network stacks. Packets leaving end hosts may be tagged with instructions for what performance metrics are required encoded in custom-designed header fields. In addition to in-network aggregation computations, switches parse and exchange information via these header fields. Ultimately packets returning to end hosts carry the desired performance data up to the CPU-based network stack where further aggregation and post processing can occur. In addition to expanding the processing flexibility and memory capacity via computation at distributed end-host CPUs, this approach also enables integrating network-stack and application-layer information into the performance monitoring results.
Key contributions. The first work in this category OmniMon [69] develops a careful method of splitting per-flow, per-switch packet count monitoring across switches and end hosts with a focus on maintaining per-epoch consistency across different observation points. In particular, they propose a hybrid consistency model (where each packet belongs to the same epoch) as a practical yet useful relaxation of strict consistency. This enables detailed detection and localization of performance events such as packet loss. LightGuardian [153], on the other hand focuses on reducing communication overheads between switches and end hosts by transmitting aggregate per-flow, per-switch performance metrics collected on switches to hosts in a novel "sketchlet" format. End hosts can incrementally aggregate received sketchlets to construct a detailed view of packet loss, latency, and jitter metrics in the network.

BufScope [54] further leverages end-host participation to attach application-layer metrics (in particular, request ids) to per-flow, per-switch metrics aggregated on switch hardware. They integrate these methods in a comprehensive system that ultimately provides queuing latency measurements for every buffer along the path of an application request.
Open problems. Although works in this section demonstrate that leveraging end-host participation in network performance monitoring has some significant advantages, it also potentially introduces new issues. In particular, the increased system complexity leads to an increased number of potential problems that could impact network performance monitoring such as end host hardware and/or software failures.

5.2 Explaining performance events

In addition to detecting the occurrence of and connections impacted by network performance anomalies, a complementary group of efforts seek to provide a basis for identifying the root-cause of such anomalies. Such root causes typically go beyond the detailed performance metrics of other works by attempting to find larger-scale correlations between performance-related events.

Common design patterns. Works in this sections trigger collection and/or reporting of telemetry data only when a performance event occurs via automatic or manual trigger conditions. Though these systems may compute metrics continuously in switches, computations and communications specific to debugging the root cause are only executed when debugging is activated. The key challenge is to strike a balance such that a minimal amount of computation overhead is required in the non-debugging state, but a maximal amount of information can be gathered and correlated when the debugging state is entered.

Key contributions. The first work in this area, SwitchPointer [132], builds on previous state-less tracing approaches [130, 131] to combine aggregation in switch hardware with path-based debugging. In particular, SwitchPointer uses switches as a "directory service": when a performance event occurs and the system enters debugging mode, the pointers stored in each switch along the problematic path are used to lookup per-flow counters maintained in relevant end hosts. Combined with a novel hierarchical division of time, this enables quick answers to questions such as which other flows were sharing a link with flow X at time t (e.g., when flow X experienced unusual queuing delay).

SpiderMon [140], on the other hand, automatically detects when flows experience high cumulative queuing latency by summing the time spent by individual packets in each queue along their network paths. When a path experiences abnormally high latency, SpiderMon triggers export of flow-level metrics along all paths intersecting the impacted path by using novel "spider" control packets exchanged between switches. A central controller receives reports generated from switches that process spider packets and (re)constructs debugging results for the impacted path.

Open problems. A key open problem in this group of works is justifying the need for extra complexity to answer questions that are theoretically already covered by simpler performance monitoring systems like FlowRadar [87]. Intuitively as networks increase in scale and complexity (e.g., total number of nodes), the overheads of always-on systems increase at a much faster rate compared to the on-demand systems considered in this section. However, it's unclear where the advantages tip from always-on to on-demand and what other aspects of the network setting may impact this tradeoff (e.g., presence of virtual routing [110], software gateways [141]).

6 NETWORK AUTOMATION

Given the large scale and complexity of modern computer networks and the need to respond to network-based performance and security events on very short timescales, a growing body of works seeks to develop data-driven solutions to automate common network operations. In this section we consider two example cases of network automation that makes direct use of programmable switch hardware to perform in-network aggregation in service of a larger network automation goal. We leave full analysis of other network automation usecases (e.g., load balancing [77, 90, 123], ML-base security classification [15, 22]) for future studies.

6.1 Volumetric DDoS Defence

Volumetric distributed denial of service (DDoS) attacks sever or degrade network connections by flooding particular network links or end hosts with large volumes of traffic sent from a large number of distributed sources [114, 115, 136, 138, 139, 142]. A variety of closed-loop systems have been proposed to defend against volumetric DDoS attacks by monitoring traffic to detect particular subsets of traffic associated with DDoS techniques and actively rate-limiting or blocking this traffic directly in programmable switch hardware.

6.1.1 **Fine-Grained, CPU-Controlled Languages.** The first group of works seeking to develop in-network DDoS defense using programmable switch hardware leverages hardware to aggregate and detect attack traffic, but also relies on a logically centralized controller (running in a CPU-based system) to coordinate deployment of attack mitigation operations in switch hardware.

Common design patterns. The works in this section propose policy languages or APIs with several primitive operations (e.g., SYN cookies, block-lists) which network administrators can use to construct DDoS mitigation mechanisms tailored to particular attack behaviors. The switch hardware implementations of these mechanisms typically use similar sketch-based approximate algorithms to deal with constrained resources.

Key contributions. Poseidon [151] introduced the first policy-language for switch hardware-based DDoS defense which includes flexible predicates to identify common attack packets (e.g., SYN packets, UDP response packets), counters to group and aggregate matching packets, and a set of threshold-based actions including dropping and rate-limiting detected attack traffic. Similar to Sonata [63], defense policies are implemented by partitioning primitive operations across programmable switch hardware and CPU-based servers. Jaqen [92]

extends the approach put forth in Poseidon by adding resource-efficient approximate switch hardware primitives tailored for particular attack vectors such as "UnmatchAndAction" for reflection-based attacks and two custom SYN-proxy designs. Ripple [144] also implements a stream-processing style mitigation policy language on programmable switch hardware, but leverages the concept of a global panoramic view of traffic (maintained by a distributed synchronization protocol between switches) to realize distributed defense against a particular class of dynamic link-flooding attacks (in particular attacks like Coremelt [129] and Crossfire [76]).

Open problems. Although Poseidon [151] and Jaqen [92] both discuss the need to dynamically reconfigure attack mitigation in the face of modern dynamic DDoS attacks, neither provides adequate mechanisms for doing so. Jaqen does include attack detection through a refined version of UnivMon [91], but it's unclear how this detection capability pairs with the proposed mitigation mechanisms. Both methods require compiling and installing defense mechanisms in switch hardware which induces several seconds of downtime during which no traffic can be processed on the switch requiring re-routing during downtime.

These methods also propose developing finely-tuned mitigation mechanisms for specific attack types, leaving open the more fundamental issue of how to defeat the inherent imbalance the underlies DDoS's effectiveness—it's very easy for the attacker to come up with a new attack vector (or modify a known vector) whereas it's very hard for a network administrator using one of the systems discussed in this section to anticipate and deploy mitigation methods for all possible attack vectors.

6.1.2 **Coarser-Grained, Switch-Only Policies.** The second group of works seeking to develop in-network DDoS defense implements all stages of attack traffic aggregation, detection, and mitigation entirely in the switch hardware pipeline. The key advantage of these approaches is the ability to quickly react to attack occurrences and changes in attack traffic without control plane involvement and/or needing to re-compile and reload new mitigation mechanisms.

Common design patterns. These works propose comparatively generic methods to both detect attack occurrences and to separate attack and benign traffic (that can still be implemented entirely in switch hardware). Due to the possibility of false positives, they propose rate-limiting the detected attack traffic rather than outright blocking or allowing as considered in the previous subsection.

Key contributions. Euclid [39] describes the first DDoS detection and mitigation method that operates entirely in a single switch hardware program. The proposed method uses sketches to estimate source and destination address entropy (following a similar design as [83]). Relative changes in entropy trigger attack detection after which Euclid applies rate limiting to the set of source, destination pairs identified as being most responsible for the change in entropy. ACCTurbo [16] describes an alternative approach to attack mitigation entirely within switch hardware by adapting the idea of aggregate-based congestion control [95]. In particular, ACCTurbo performs simple clustering in switch hardware and leverages a CPU-based control plane to install rate-limiting rules on large aggregates that may be associated with attack traffic.

Open problems. Although the works considered in this section effectively develop DDoS mitigation approaches that can function at runtime without interrupting packet forwarding, they do not offer the same level of precision as Poseidon [151] or Jaqen [92] or a path towards distributed mitiagtion as in Ripple [144]. A key open question then is how to improve the precision of these methods while still working in the confines of switch hardware.

Additionally, the design pattern of maintaining all attack mitigation related state entirely in the data plane has a side effect of preventing network administrators from observing the mitigation system's actions (e.g., to ensure mission-critical traffic is not accidentally blocked by mitigation). Since Euclid acts on a packet-by-packet level and does not store particular source and destination addresses, it is literally impossible to assess what traffic it will rate-limit or block. ACCTurbo improves on this situation somewhat by exposing aggregate-level information to the control plane, but does not offer motivation for why particular aggregates should be considered attack traffic—in particular, the aggregates reported by ACCTurbo may contain a mix of both attack and benign traffic since switch resource limitations require relatively coarse-grained aggregates compared to the source, destination level considered in Euclid. Overall, these observations motivate an additional open problem of how to balance interpretability of decisions made by these types of defense systems with sufficient precision of mitigation actions.

6.2 Flow Offloading

Certain types of network processing platforms (e.g., cloud gateways [110]) divide traffic processing between hardware and software platforms in order to leverage both the larger memory available in software as well as the higher throughput available in hardware [120]. These systems monitor traffic to automatically select a subset of network flows which are most advantageous to offload to a limited-capacity, high-throughput hardware platform. The traffic monitoring task driving such systems aggregates packets into flows (e.g., based on fivetuple) and reports the top-n flows with the highest packet rates.

Common design patterns. These works place programmable switch hardware in front of the CPU-based routing system. Over time, certain heavy and stable flows or table entries (e.g., routing policies) are identified in the CPU system and added to switch hardware to process traffic for these flows or entries before their packets reach the CPU. Hardware also keeps track of utilization (via simple counters) of each offloaded entry and reports these counters to the CPU to help prioritize which hardware entries to move back to the CPU when they terminate or become less heavy.

Key contributions. Sailfish [110] motivates the need for hybrid hardware, software design of cloud gateways and presents several mechanisms (e.g., mapping large tables across multiple switch pipelines) to maximize the number of flows which can be processed in switch hardware. Elixir [141] extends this hybrid design by developing novel algorithms targeted specifically at efficiently detecting and offloading bursty flows, which have a disproportionately negative impact on CPU performance.

Open problems. Works so far in this area inherit the "scale via adding more CPU cores" idea which made the CPU-based cloud gateway idea originally feasible. However, there's still a question of how the monitoring capabilities in switch hardware could be used to further reduce CPU resource requirements. For example, hardware could help with the top-n and bottom-n detection problems required to make decisions about which flows / entries should be moved between hardware and software.

7 SUMMARY & FUTURE OUTLOOK

The research efforts discussed in this work trace the ramifications of programmable switch hardware on how network administrators extract critical insights about the traffic flowing through their networks over the last 10 years. We discussed how a wide range of metrics have been proposed and implemented for various traffic profiling purposes (§ 4), how detailed traffic monitoring enables fast detection and explanation of performance degradations in data center networks (§ 5), and how specific monitoring data has been incorporated in two particular end-to-end network automation tasks (§ 6). In each of these application areas, we observed the different types of aggregation computations considered as well as the particular classes of algorithms (e.g., sketches [36]) and techniques for realizing implementation of these algorithms in systems centered around programmable switch hardware.

We also observed that several open research problems remain in each of the distinct categories of works considered. Looking across all of these categories, we distill these open problems into three high-level ideas for future research.

Idea 1: careful reconsideration of sketch-based approaches. The simple per-packet update procedures, flexibility to estimate a wide range of different aggregation computations, and existence of rigorous statistical analysis of error bounds leads to sketch-based algorithms playing a significant roll in traffic monitoring literature. However, from a practical application perspective, sketches are fundamentally limited in the dependence of their accuracy on underlying traffic properties (e.g., the number of observed flow keys). We anticipate developing novel methods to the aggregation computations required by network traffic monitoring applications that can execute in constrained switch hardware while also adapting to differing traffic characteristics (e.g., changes in the number of flow keys observed) will be a critical direction for future research. These methods may be developed through extension of current sketch-based techniques or may require considering different forms of approximation (e.g., cluster sampling as used in DynATOS [102]).

Idea 2: monitoring tasks should be driven by real-life concerns like network automation problems. Although works like Sonata [63], DynATOS [102], and SketchLib [106] implement systems that can simultaneously compute a wide range of traffic metrics, it is not always clear which metrics are most critical for network administrators and how to design interfaces to actually support a wide range of monitoring tasks (e.g., multiple network automation systems). To illustrate this disconnect, consider the 11 monitoring queries proposed in Sonata and subsequently used to evaluate several other works [69, 86, 102, 157]. Although these queries effectively demonstrate the possibilities of the stream-processing type of interface, it's unclear how they relate to actual network automation requirements (e.g., the types of metrics used by DDoS defense systems from § 6.1). We argue that the relatively recent burst of research activity in concrete network automation applications (e.g., all works considered in § 6 are published during or after 2020) represents the leading edge of a much richer body of future research into these kinds of concrete applications. Once a sufficient body of such examples have been considered, perhaps the question of how to design a generic interface and implementation for network traffic monitoring can be revisited to much greater effect.

Idea 3: improved data sets for evaluation. A key practical challenge faced by traffic monitoring research in general is the sparsity of publicly-available data sets to use in evaluating proposed methods and techniques. This issue is amplified in switch hardware based systems due the fact that switches ultimately are expected to process extremely high data rates (e.g., 2 Tbps) and have a central position in network topologies (e.g., traffic might come from one of hundreds of ports). Since switch processing operates at a per-packet level, evaluation of switch-based systems requires packet-level traces or

traffic generation techniques. Moreover, these traces should contain relevant types of traffic for the network settings considered (e.g., ISPs, data centers). Additionally, works that seek to detect certain traffic events require traces that contain (ideally a large number of) relevant events combined with normal background traffic.

Many of the traffic profiling works discussed in previous sections use some combination of two publicly available packet-level traffic datasets: the CAIDA Internet traces [3] and MAWILab [51]. These data sets both come from single links in large ISP networks and only provide between 15 minutes and one hour worth of data at limited data rates (around 5 Gbps) compared to switch throughput. While these traces provide useful confirmation of a system's ability to handle certain types of traffic and can be sped up to closer approach realistic workloads in multi-port switches (as in Sonata's [63] evaluation), in many cases it's unclear if they actually contain the types of events systems are seeking to detect (e.g., the security-related events considered in Sonata [63]) The situation is even more challenging in network automation works where precise types of traffic scenarios are considered and decisions by the network automation system may directly impact observed traffic behavior. As a result, most of the DDoS defense works described in § 6.1 use simplistic synthetic traffic generation methods (e.g., using MoonGen [46]) or outdated public data sets [4, 117] whereas the flow offloading works described in § 6.2 primarily rely on private traces from sponsoring organization's data centers. Given the vested interest of data owners (e.g., cloud service providers, ISPs) in maintaining confidentiality of customer and internal information, solutions to this problem will likely require significant creative effort from the research community (see for example [85]).

REFERENCES

[1] [n. d.]. Barefoot Networks - Wikipedia. https://en.wikipedia.org/wiki/Barefoot_Networks. ([n. d.]). Accessed: 2023-03-05.

[2] [n. d.]. Behavioral Model Repository. https://github.com/p4lang/behavioral-model. ([n. d.]). Accessed: 2023-03-05.

[3] [n. d.]. The CAIDA UCSD Anonymized Internet Traces Dataset - 2019. https://www.caida.org/data/monitors/passive-equinix-nyc.xml. ([n. d.]).

[4] [n. d.]. The CAIDA UCSD "DDoS Attack 2007" Dataset. https://www.caida.org/catalog/datasets/ddos-20070804_dataset. ([n. d.]). Accessed: 2023-03-08.

[5] [n. d.]. Intel Exits Another Non-Core Business. https://www.fool.com/investing/2023/01/29/intel-exits-another-non-core-business/. ([n. d.]). Accessed: 2023-03-05.

[6] [n. d.]. Intel Tofino Series. https://www.intel.com/content/www/us/en/products/details/network-io/programmable-ethernet-switch/tofino-series.html. ([n. d.]). Accessed: 2023-02-20.

[7] [n. d.]. Intel's Datacenter Business Goes from Bad to Worse, With Worst Still to Come. https://www.nextplatform.com/2023/01/27/intels-datacenter-business-goes-from-bad-to-worse-with-worst-still-to-come/. ([n. d.]). Accessed: 2023-03-05.

[8] [n. d.]. NetFPGA. https://netfpga.org/. ([n. d.]). Accessed: 2023-03-05.

[9] [n. d.]. NPL-Open, High-level language for developing feature-rich solutions for programmable networking platforms. https://nplang.org/. ([n. d.]). Accessed: 2023-03-08.

[10] [n. d.]. NSDI '22 | USENIX. https://www.usenix.org/conference/nsdi22. ([n. d.]). Accessed: 2023-03-08.

[11] [n. d.]. Traffic monitoring using sFlow. https://sflow.org/sFlowOverview.pdf. ([n. d.]). Accessed: 2023-02-20.

[12] [n. d.]. Welcome | acm sigcomm. https://sigcomm.org/events/sigcomm-conference. ([n. d.]). Accessed: 2023-03-08.

[13] [n. d.]. The Zeek Network Security Monitor. https://zeek.org/. ([n. d.]). Accessed December 2022.

[14] Anup Agarwal, Zaoxing Liu, and Srinivasan Seshan. 2022. HeteroSketch: Coordinating Network-wide Monitoring in Heterogeneous and Dynamic Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 719–741.

[15] Aristide Tanyi-Jong Akem, Michele Gucciardo, Marco Fiore, et al. 2023. Flowrest: Practical Flow-Level Inference in Programmable Switches with Random Forests. In *IEEE International Conference on Computer Communications*.

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

[17] Ali AlSabeh, Joseph Khoury, Elie Kfoury, Jorge Crichigno, and Elias Bou-Harb. 2022. A survey on security applications of p4 programmable switches and a stride-based vulnerability assessment. *Computer Networks* 207 (2022), 108800.

[18] Anoud Alshnakat, Didrik Lundberg, Roberto Guanciale, Mads Dam, and Karl Palmskog. 2022. HOL4P4: semantics for a verified data plane. In *Proceedings of the 5th International Workshop on P4 in Europe*. 39–45.

[19] Nikos Anerousis, Prosper Chemouil, Aurel A Lazar, Nelu Mihai, and Stephen B Weinstein. 2021. The origin and evolution of open programmable networks and SDN. *IEEE Communications Surveys & Tutorials* 23, 3 (2021), 1956–1971.

[20] Behnaz Arzani, Selim Ciraci, Luiz Chamon, Yibo Zhu, Hongqiang Harry Liu, Jitu Padhye, Boon Thau Loo, and Geoff Outhred. 2018. 007: Democratically finding the cause of packet drops. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 419–435.

[21] Tom Barbette, Chen Tang, Haoran Yao, Dejan Kostic, Gerald Q Maguire Jr, Panagiotis Papadimitratos, and Marco Chiesa. 2020. A High-Speed Load-Balancer Design with Guaranteed Per-Connection-Consistency.. In *NSDI*. 667–683.

[22] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. 2021. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications.. In *NDSS*.

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

[24] Rohan Bhatia, Arpit Gupta, Rob Harrison, Daniel Lokshtanov, and Walter Willinger. 2021. DynamiQ: Planning for dynamics in network streaming analytics systems. *arXiv preprint arXiv:2106.05420* (2021).

[25] Roberto Bifulco and Gábor Rétvári. 2018. A survey on the programmable data plane: Abstractions, architectures, and open problems. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 1–7.

[26] Kevin Borders, Jonathan Springer, and Matthew Burnside. 2012. Chimera: A declarative language for streaming network traffic analysis. In *Proceedings of the USENIX Security Symposium*. 365–379.

[27] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[28] 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. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 99–110.

[29] Vladimir Braverman and Rafail Ostrovsky. 2010. Zero-one frequency laws. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 281–290.

[30] Vladimir Braverman and Rafail Ostrovsky. 2013. Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*. Springer, 58–70.

[31] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *Automata, Languages and Programming: 29th International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002 Proceedings 29*. Springer, 693–703.

[32] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, and Ori Rottenstreich. 2018. Catching the microburst culprits with snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*. 22–28.

[33] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. 2020. BeauCoup: Answering Many Network Traffic Queries, One Memory Update at a Time. In *Proceedings of the conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 226–239.

[34] Francesco Ciaccia, Ivan Romero, Oriol Arcas-Abella, Diego Montero, René Serral-Gracià, and Mario Nemirovsky. 2020. Sabes: Statistical available bandwidth estimation from passive tcp measurements. In *2020 IFIP Networking Conference (Networking)*. IEEE, 743–748.

[35] Benoit Claise. 2004. *Cisco systems netflow services export version 9*. Technical Report.

[36] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[37] Paolo Costa, Austin Donnelly, Antony IT Rowstron, and Greg O'Shea. 2012. Camdoop: Exploiting In-network Aggregation for Big Data Applications.. In *NSDI*, Vol. 12. 3–3.

[38] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. 2003. Gigascope: a stream database for network applications. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 647–651.

[39] Alexandre da Silveira Ilha, Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. 2020. Euclid: A fully in-network, p4-based approach for real-time ddos attack detection and mitigation. *IEEE Transactions on Network and Service Management* (2020).

[40] Bruno L Dalmazo, Jonatas A Marques, Lucas R Costa, Michel S Bonfim, Ranyelson N Carvalho, Anderson S da Silva, Stenio Fernandes, Jacir L Bordim, Eduardo Alchieri, Alberto Schaeffer-Filho, et al. 2021. A systematic review on distributed denial of service attack defense mechanisms in programmable networks. *International Journal of Network Management* 31, 6 (2021), e2163.

[41] Huynh Tu Dang, Marco Canini, Fernando Pedone, and Robert Soulé. 2016. Paxos made switch-y. *ACM SIGCOMM Computer Communication Review* 46, 2 (2016), 18–24.

[42] Tooska Dargahi, Alberto Caponi, Moreno Ambrosin, Giuseppe Bianchi, and Mauro Conti. 2017. A survey on the security of stateful SDN data planes. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1701–1725.

[43] J Davin, JD Case, M Fedor, and ML Schoffstall. 1990. *Simple network management protocol (SNMP)*. RFC 1157.

[44] Ryan Doenges, Mina Tahmasbi Arashloo, Santiago Bautista, Alexander Chang, Newton Ni, Samwise Parkinson, Rudy Peterson, Alaia Solko-Breslin, Amanda Xu, and Nate Foster. 2021. Petr4: formal foundations for p4 data planes. *Proceedings of the ACM on Programming Languages* 5, POPL (2021).

[45] Svante Ekelin, Martin Nilsson, Erik Hartikainen, Andreas Johnsson, J-E Mangs, Bob Melander, and Mats Bjorkman. 2006. Real-time measurement of end-to-end available bandwidth using kalman filtering. In *2006 ieee/ifip network operations and management symposium noms 2006*. IEEE, 73–84.

[46] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Internet Measurement Conference 2015 (IMC'15)*. Tokyo, Japan.

[47] David Eppstein, Michael T Goodrich, Frank Uyeda, and George Varghese. 2011. What's the difference? Efficient set reconciliation without prior context. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 218–229.

[48] Cristian Estan, Stefan Savage, and George Varghese. 2003. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. 137–148.

[49] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2014. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review* 44, 2 (2014), 87–98.

[50] Rodrigo Fonseca, Tianrong Zhang, Karl Deng, and Lihua Yuan. 2019. dShark: A general, easy to program and scalable framework for analyzing in-network packet traces. (2019).

[51] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. 2010. MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. In *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 12.

[52] Martino Fornasa and Massimo Maresca. 2011. Passive Access Capacity Estimation through the Analysis of Packet Bursts. In *Access Networks: 5th International ICST Conference on Access Networks, AccessNets 2010 and First ICST International Workshop on Autonomic Networking and Self-Management in Access Networks, SELFMAGICNETS 2010, Budapest, Hungary, November 3-5, 2010, Revised Selected Papers 5*. Springer, 83–99.

[53] Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. 2011. Frenetic: A network programming language. *ACM Sigplan Notices* 46, 9 (2011), 279–291.

[54] Kaihui Gao, Chen Sun, Shuai Wang, Dan Li, Yu Zhou, Hongqiang Harry Liu, Lingjun Zhu, and Ming Zhang. 2022. Buffer-based End-to-end Request Event Monitoring in the Cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 829–843.

[55] Ya Gao and Zhenling Wang. 2021. A review of P4 programmable data planes for network security. *Mobile Information Systems* 2021 (2021), 1–24.

[56] Nadeen Gebara, Manya Ghobadi, and Paolo Costa. 2021. In-network aggregation for shared machine learning clusters. *Proceedings of Machine Learning and Systems* 3 (2021), 829–844.

[57] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. 2019. SIMON: A simple and scalable method for sensing, inference and measurement in data center networks. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 549–564.

[58] Phillip B Gibbons. 2001. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB*, Vol. 1. 541–550.

[59] Jose Gomez, Elie F Kfoury, Jorge Crichigno, and Gautam Srivastava. 2022. A survey on TCP enhancements using P4-programmable devices. *Computer Networks* 212 (2022), 109030.

[60] Michael T Goodrich and Michael Mitzenmacher. 2011. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 792–799.

[61] Richard L Graham, Devendar Bureddy, Pak Lui, Hal Rosenstock, Gilad Shainer, Gil Bloch, Dror Goldenerg, Mike Dubman, Sasha Kotchubievsky, Vladimir Koushnir, et al. 2016. Scalable hierarchical aggregation protocol (SHArP): a hardware architecture for efficient data reduction. In *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*. IEEE, 1–10.

[62] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. 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*. 139–152.

[63] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: Query-driven streaming network telemetry. In *Proceedings of the conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 357–371.

[64] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. 2018. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research*. 1–7.

[65] Frederik Hauser, Marco Häberle, Daniel Merling, Steffen Lindner, Vladimir Gurevich, Florian Zeiger, Reinhard Frank, and Michael Menth. 2022. A survey on data plane programming with p4: Fundamentals, advances, and applied research. *Journal of Network and Computer Applications* (2022), 103561.

[66] Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology*. 683–692.

[67] Qun Huang, Patrick PC Lee, and Yungang Bao. 2018. Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 576–590.

[68] Qun Huang, Siyuan Sheng, Xiang Chen, Yungang Bao, Rui Zhang, Yanwei Xu, and Gong Zhang. 2021. Toward Nearly-Zero-Error Sketching via Compressive Sensing.. In *NSDI*. 1027–1044.

[69] Qun Huang, Haifeng Sun, Patrick PC Lee, Wei Bai, Feng Zhu, and Yungang Bao. 2020. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 404–421.

[70] Nikita Ivkin, Edo Liberty, Kevin Lang, Zohar Karnin, and Vladimir Braverman. 2022. Streaming quantiles algorithms with small space and update time. *Sensors* 22, 24 (2022), 9612.

[71] Nikita Ivkin, Zhuolong Yu, Vladimir Braverman, and Xin Jin. 2019. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 285–291.

[72] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 121–136.

[73] Lavanya Jose and Minlan Yu. 2011. Online measurement of large traffic aggregates on commodity switches. In *Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 11)*.

[74] Raj Joshi, Ting Qu, Mun Choon Chan, Ben Leong, and Boon Thau Loo. 2018. BurstRadar: Practical real-time microburst monitoring for datacenter networks. In *Proceedings of the 9th Asia-Pacific Workshop on Systems*. 1–8.

[75] Enio Kaljic, Almir Maric, Pamela Njemcevic, and Mesud Hadzialic. 2019. A survey on data plane flexibility and programmability in software-defined networking. *IEEE Access* 7 (2019), 47804–47840.

[76] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. 2013. The crossfire attack. In *2013 IEEE symposium on security and privacy*. IEEE, 127–141.

[77] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*. 1–12.

[78] Elie F Kfoury, Jorge Crichigno, and Elias Bou-Harb. 2021. An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access* 9 (2021), 87094–87155.

[79] Faisal Khan, Nicholas Hosein, Soheil Ghiasi, Chen-Nee Chuah, and Puneet Sharma. 2013. Streaming solutions for fine-grained network traffic measurements and analysis. *IEEE/ACM Transactions On Networking* 22, 2 (2013), 377–390.

[80] Anurag Khandelwal, Rachit Agarwal, and Ion Stoica. 2019. Confluo: Distributed Monitoring and Diagnosis Stack for High-speed Networks.. In *NSDI*. 421–436.

[81] Ali Kheradmand and Grigore Rosu. 2018. P4K: A formal semantics of P4 and applications. *arXiv preprint arXiv:1804.01468* (2018).

[82] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. 2020. Tea: Enabling state-intensive network functions on 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*. 90–106.

[83] Yu-Kuen Lai, Ku-Yeh Shih, Po-Yu Huang, Ho-Ping Lee, Yu-Jau Lin, Te-Lung Liu, and Jim Hao Chen. 2019. Sketch-based entropy estimation for network traffic analysis using programmable data plane ASICs. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 1–2.

[84] ChonLam Lao, Yanfang Le, Kshiteej Mahajan, Yixi Chen, Wenfei Wu, Aditya Akella, and Michael M Swift. 2021. ATP: In-network Aggregation for Multi-tenant Learning.. In *NSDI*, Vol. 21. 741–761.

[85] Yukhe Lavinia, Ramakrishnan Durairajan, Reza Rejaie, and Walter Willinger. 2020. Challenges in using ML for networking research: How to label if you must. In *Proceedings of the Workshop on Network Meets AI & ML*. 21–27.

[86] Yiran Li, Kevin Gao, Xin Jin, and Wei Xu. 2020. Concerto: cooperative network-wide telemetry with controllable error rate. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*. 114–121.

[87] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: a better NetFlow for data centers. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 311–324.

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

[89] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. 2017. Incbricks: Toward in-network computation with an in-network cache. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. 795–809.

[90] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. 2019. DistCache: Provable Load Balancing for Large-Scale Storage Systems with Distributed Caching.. In *FAST*, Vol. 19. 143–157.

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

[92] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches. In *30th USENIX Security Symposium (USENIX Security 21)*.

[93] Zaoxing Liu, Samson Zhou, Ori Rottenstreich, Vladimir Braverman, and Jennifer Rexford. 2020. Memory-efficient performance monitoring on programmable switches with lean algorithms. In *Symposium on Algorithmic Principles of Computer Systems*. SIAM, 31–44.

[94] Liang Luo, Jacob Nelson, Luis Ceze, Amar Phanishayee, and Arvind Krishnamurthy. 2018. Parameter hub: a rack-scale parameter server for distributed deep neural network training. In *Proceedings of the ACM Symposium on Cloud Computing*. 41–54.

[95] Ratul Mahajan, Steven M Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. 2002. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review* 32, 3 (2002), 62–73.

[96] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. 2006. Is sampled data sufficient for anomaly detection?. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 165–176.

[97] Luo Mai, Lukas Rupprecht, Abdul Alim, Paolo Costa, Matteo Migliavacca, Peter Pietzuch, and Alexander L Wolf. 2014. Netagg: Using middleboxes for application-specific on-path aggregation in data centres. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 249–262.

[98] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review* 38, 2 (2008), 69–74.

[99] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *Database Theory-ICDT 2005: 10th International Conference, Edinburgh, UK, January 5-7, 2005. Proceedings 10*. Springer, 398–412.

[100] Oliver Michel, Roberto Bifulco, Gabor Retvari, and Stefan Schmid. 2021. The programmable data plane: Abstractions, architectures, algorithms, and applications. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–36.

[101] Gregory T Minton and Eric Price. 2014. Improved concentration bounds for count-sketch. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 669–686.

[102] Chris Misa, Walt O'Connor, Ramakrishnan Durairajan, Reza Rejaie, and Walter Willinger. 2022. Dynamic Scheduling of Approximate Telemetry Queries. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 701–717.

[103] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2014. DREAM: Dynamic resource allocation for software-defined measurement. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 419–430.

[104] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2015. SCREAM: Sketch resource allocation for software-defined measurement. In *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 14.

[105] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. 2016. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. 129–143.

[106] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. 2022. SketchLib: Enabling Efficient Sketch-based Monitoring on Programmable Switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 743–759.

[107] Jad Naous, Glen Gibb, Sara Bolouki, and Nick McKeown. 2008. NetFPGA: reusable router architecture for experimental research. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. 1–7.

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

[109] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122–144.

[110] Tian Pan, Nianbing Yu, Chenhao Jia, Jianwen Pi, Liang Xu, Yisong Qiao, Zhiguo Li, Kun Liu, Jie Lu, Jianyuan Lu, et al. 2021. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 194–206.

[111] Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks* 31, 23-24 (1999), 2435–2463.

[112] Yongyi Ran, Xiaoban Wu, Peilong Li, Chen Xu, Yan Luo, and Liang-Min Wang. 2018. Equery: Enable event-driven declarative queries in programmable network measurement. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–7.

[113] Marshall T Rose and Keith McCloghrie. 1990. *Structure and Identification of Management Information for TCP/IP-based internets*. RFC 1155.

[114] Christian Rossow. 2014. Amplification Hell: Revisiting Network Protocols for DDoS Abuse.. In *NDSS*. 1–15.

[115] Lauren Rudman and B Irwin. 2015. Characterization and analysis of NTP amplification based DDoS attacks. In *2015 Information Security for South Africa (ISSA)*. IEEE, 1–5.

[116] Fabian Ruffy, Tao Wang, and Anirudh Sivaraman. 2020. Gauntlet: Finding bugs in compilers for programmable packet processing. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*. 683–699.

[117] José Jair Santanna, Roland van Rijswijk-Deij, Rick Hofstede, Anna Sperotto, Mark Wierbosch, Lisandro Zambenedetti Granville, and Aiko Pras. 2015. Booters—An analysis of DDoS-as-a-service attacks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 243–251.

[118] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. 2017. In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. 150–156.

[119] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. 2019. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701* (2019).

[120] Nadi Sarrar, Steve Uhlig, Anja Feldmann, Rob Sherwood, and Xin Huang. 2012. Leveraging Zipf's law for traffic offloading. *ACM SIGCOMM Computer Communication Review* 42, 1 (2012), 16–22.

[121] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. 2022. Continuous in-network round-trip time monitoring. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 473–485.

[122] Mohamed A Sharaf, Jonathan Beaver, Alexandros Labrinidis, and Panos K Chrysanthis. 2003. TiNA: A scheme for temporal coherency-aware in-network aggregation. In *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*. 69–76.

[123] Naveen Kr Sharma, Antoine Kaufmann, Thomas E Anderson, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. 2017. Evaluating the Power of Flexible Packet Processing for Network Resource Allocation.. In *NSDI*. 67–82.

[124] Anirudh Sivaraman, Alvin Cheung, Mihai Budiu, Changhoon Kim, Mohammad Alizadeh, Hari Balakrishnan, George Varghese, Nick McKeown, and Steve Licking. 2016. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 15–28.

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

[126] John Sonchack, Adam J Aviv, Eric Keller, and Jonathan M Smith. 2018. Turboflow: Information rich flow record generation on commodity switches. In *Proceedings of the Thirteenth EuroSys Conference*. 1–16.

[127] 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 *2018 USENIX Annual Technical Conference (USENIXATC 18)*. 823–835.

[128] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. 2020. FCM-sketch: generic network measurements with data plane support. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 78–92.

[129] Ahren Studer and Adrian Perrig. 2009. The coremelt attack. In *European Symposium on Research in Computer Security*. Springer, 37–52.

[130] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2015. Cherrypick: Tracing packet trajectory in software-defined datacenter networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. 1–7.

[131] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2016. Simplifying Datacenter Network Debugging with PathDump. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 233–248.

[132] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. 2018. Distributed network monitoring and debugging with SwitchPointer. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 453–456.

[133] Lu Tang, Qun Huang, and Patrick PC Lee. 2020. SpreadSketch: Toward invertible and network-wide detection of superspreaders. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 1608–1617.

[134] Ross Teixeira, Rob Harrison, Arpit Gupta, and Jennifer Rexford. 2020. PacketScope: Monitoring the packet lifecycle inside a switch. In *Proceedings of the Symposium on SDN Research*. 76–82.

[135] Daniel Ting. 2018. Data sketches for disaggregated subset sum and frequent item estimation. In *Proceedings of the 2018 International Conference on Management of Data*. 1129–1140.

[136] Alethea Toh, Anupam Vij, and Syed Pasha. [n. d.]. Azue DDoS Protection—2021 Q3 and Q4 DDoS attack trends. https://tinyurl.com/45uwpjem. ([n. d.]). Accessed: 2022.

[137] Gerry Wan, Fengchen Gong, Tom Barbette, and Zakir Durumeric. 2022. Retina: analyzing 100GbE traffic on commodity hardware. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 530–544.

[138] An Wang, Wentao Chang, Songqing Chen, and Aziz Mohaisen. 2018. Delving into internet DDoS attacks by botnets: characterization and analysis. *IEEE/ACM Transactions on Networking* 26, 6 (2018), 2843–2855.

[139] An Wang, Aziz Mohaisen, and Songqing Chen. 2017. An adversary-centric behavior modeling of DDoS attacks. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1126–1136.

[140] Weitao Wang, Xinyu Crystal Wu, Praveen Tammana, Ang Chen, and TS Eugene Ng. 2022. Closed-loop Network Performance Monitoring and Diagnosis with SpiderMon. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 267–285.

[141] Yanshu Wang, Dan Li, Yuanwei Lu, Jianping Wu, Hua Shao, and Yutian Wang. 2022. Elixir: A High-performance and Low-cost Approach to Managing Hardware/Software Hybrid Flow Tables Considering Flow Burstiness. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 535–550.

[142] Arne Welzel, Christian Rossow, and Herbert Bos. 2014. On measuring the impact of DDoS botnets. In *Proceedings of the Seventh European Workshop on System Security*. 1–6.

[143] Qingjun Xiao, Zhiying Tang, and Shigang Chen. 2020. Universal online sketch for tracking heavy hitters and estimating moments of data streams. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 974–983.

[144] Jiarong Xing, Wenqing Wu, and Ang Chen. 2021. Ripple: A Programmable, Decentralized Link-Flooding Defense Against Adaptive Adversaries. In *30th USENIX Security Symposium (USENIX Security 21)*.

[145] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 561–575.

[146] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 29–42.

[147] Zhuolong Yu, Yiwen Zhang, Vladimir Braverman, Mosharaf Chowdhury, and Xin Jin. 2020. Netlock: Fast, centralized lock management using 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*. 126–138.

[148] Lihua Yuan, Chen-Nee Chuah, and Prasant Mohapatra. 2007. ProgME: towards programmable network measurement. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. 97–108.

[149] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. 2017. Quantitative network monitoring with NetQRE. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 99–112.

[150] Kaiyuan Zhang, Danyang Zhuo, and Arvind Krishnamurthy. 2020. Gallium: Automated software middlebox offloading to 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*. 283–295.

[151] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)*.

[152] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. 2021. CocoSketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 207–222.

[153] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. 2021. LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets.. In *NSDI*. 991–1010.

[154] Hao Zheng, Chen Tian, Tong Yang, Huiping Lin, Chang Liu, Zhaochen Zhang, Wanchun Dou, and Guihai Chen. 2022. FlyMon: enabling on-the-fly task reconfiguration for network measurement. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 486–502.

[155] Yu Zhou, Jun Bi, Tong Yang, Kai Gao, Jiamin Cao, Dai Zhang, Yangyang Wang, and Cheng Zhang. 2020. Hypersight: Towards scalable, high-coverage, and dynamic network monitoring queries. *IEEE Journal on Selected Areas in Communications* 38, 6 (2020), 1147–1160.

[156] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. 2020. 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*. 76–89.

[157] Yu Zhou, Dai Zhang, Kai Gao, Chen Sun, Jiamin Cao, Yangyang Wang, Mingwei Xu, and Jianping Wu. 2020. Newton: Intent-driven network traffic monitoring. In *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. 295–308.

[158] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. 2015. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 479–491.