Stats 101 in P4: Towards In-Switch Anomaly Detection

Sam Gao
University College London
London, UK
sam.gao.17@ucl.ac.uk

Mark Handley
University College London
London, UK
m.handley@ucl.ac.uk

Stefano Vissicchio
University College London
London, UK
s.vissicchio@ucl.ac.uk

ABSTRACT

Data plane programmability is greatly improving network monitoring. Most new proposals rely on controllers pulling information (e.g., sketches or packets) from the data plane. This architecture is not a good fit for tasks requiring high reactivity, such as failure recovery, attack mitigation, and so on. Focusing on these tasks, we argue for a different architecture, where the data plane autonomously detects anomalies and pushes alerts to the controller. As a first step, we demonstrate that statistical checks can be implemented in P4 by revisiting definition and online computation of statistical measures. We collect our techniques in a P4 library, and showcase how they enable in-switch anomaly detection.

ACM Reference Format:
Sam Gao, Mark Handley, and Stefano Vissicchio. 2021. Stats 101 in P4: Towards In-Switch Anomaly Detection. In *The Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21), November 10–12, 2021, Virtual Event, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3484266.3487370

1 INTRODUCTION

Programmable data planes [1, 2] enable improvements in network monitoring; traditional switches and routers only expose a fixed set of monitoring functions (e.g., [5, 8]), which forces the architecture displayed in Figure 1a to be adopted. Programmability allow operators to decide what data to monitor at every switch, potentially for every single packet.

Most prior work assumes the architecture shown in Figure 1b, where switches are instructed to store custom sketches from which the controller extracts traffic data. We denote this architecture as *sketch-only*, since the data plane only maintains sketches. This approach has been proved effective for several tasks, including tracking heavy prefixes, number and sizes of flows, super-spreaders, packet losses and network performance [4, 10, 15, 16, 18, 19, 23, 24, 28]. QPipe [13] also explores estimating quantiles in sketches.

While very flexible, the sketch-only architecture does not achieve high reactivity. To quickly react to possibly rare events, the controller would need to pull sketches from switches every few milliseconds, which produces high overhead throughout normal operation, and may simply not be supported by the network (e.g., because of switch-to-controller delays) or the devices (e.g., because reading thousands of registers takes several milliseconds).

Fundamentally, for any sketch-only system, a delay is inevitable between when a traffic change is theoretically detectable and when the system is actually able to detect the change: this delay is inversely proportional to the generated overhead, and constrained by network characteristics, such as link delays and switches' memory access speed. In-band telemetry [20] and packet sampling [21] can be seen as degenerate cases of sketch-only systems where minimal sketches (e.g., packet headers) are sent to the controller for every monitored packet: they incur the above limitation too.

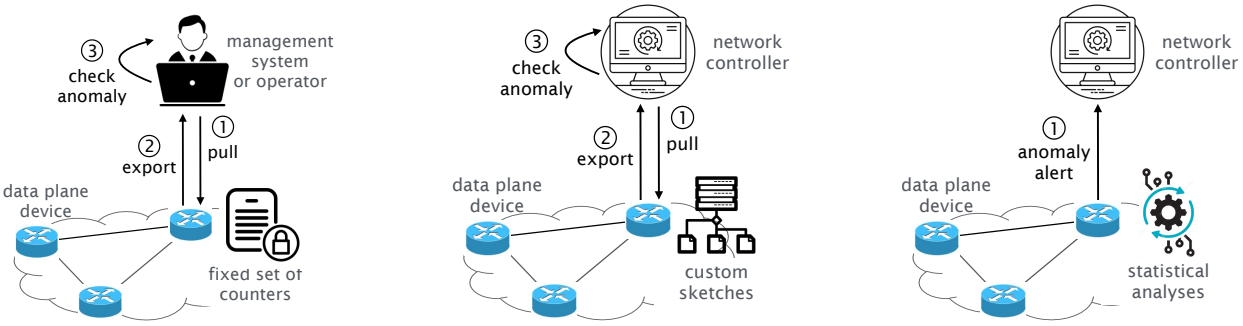
Yet, for many tasks very fast reactivity matters! An example is failure reaction, as also testified by the still ongoing industrial efforts to minimize failures' impact in traditional networks (e.g., [3, 17]). Other examples include DDoS protection (especially given that attackers can generate Tbps of traffic nowadays [6]), load balancing (e.g., to avoid servers being overloaded), and even newer machine learning applications (e.g., to avoid traffic misclassification due to outdated models in the switches [27]). Table 1 recaps those use cases.

For this class of very time-sensitive tasks, we argue for the monitoring architecture shown in Figure 1c, where anomaly detection is performed *within* programmable switches. This enables switches to both locally react to anomalies (e.g., rate limiting some flows or rerouting packets) and notify the controller for longer-term reaction (e.g., triggering additional traffic analysis and orchestrating network-wide reaction).

Supporting our envisioned approach is not straightforward though. Quickly detecting anomalies on large volumes

HotNets '21, November 10–12, 2021, Virtual Event, United Kingdom

ACM ISBN 978-1-4503-9087-3/21/11…$15.00
https://doi.org/10.1145/3484266.3487370



(a) Traditional approach. **(b) Sketch-only approach.** **(c) Envisioned approach.**

Figure 1: We argue for an architectural shift of network anomaly-detection systems, where in-switch detection algorithms are used to react to anomalies or trigger additional analyses in the controller.

of live traffic is hard per se. For data-plane-only techniques, challenges are magnified by the scarce resources and fundamental limitations of programmable switches, including lack of support for division and eliminating loops to bound packet-processing time. This is probably why only a few contributions rely on in-switch anomaly detection [12, 29], and they use basic algorithms such as thresholding to detect specific anomalies like packet loss.

We then ask ourselves if programmable switches can implement *generic* anomaly detection algorithms. Namely, we investigate if online statistical analyses can become a new data plane primitive, instrumental to anomaly detection.

This paper represents a first step in this space; we find that we can indeed support simple but generic statistical analysis in P4. We present techniques to track mean, variance, standard deviation and percentiles of distributions of arbitrary values (e.g., traffic volumes over time, packet rates per prefix, frequency of SYN packets and so on). This for example enables easy detection of outliers in normal distributions. We address the challenges of limited resources and expressivity of P4 switches by: (i) tweaking definitions of statistical measures and online algorithms to compute them, so as to avoid operations not supported by P4 (Sec. 2); and (ii) carefully engineering the use of match-action tables and registers, e.g., to avoid heavy operations such as recirculation (Sec. 3).

We pack our techniques in a P4 library, called Stat4, that also supports tuning the tracked distributions at runtime, without recompiling the P4 program. We test Stat4's correctness in the P4 behavioral model (`bmv2`), implement an anomaly detection application on top of our library, and evaluate the resource consumption of this application[1] (Sec. 3-4). Our library implementation is available at [9].

[1] we evaluate resource consumption on an application built on top of Stat4 because consumed resources depend both on the Stat4's internals and on how Stat4's functions and tables are actually used.

use case	motivation	values of interest X
remote failure	satisfy uptime SLAs	stalled flows over time
volumetric DDoS	protect network	traffic rate over time
SYN flood	protect servers	SYN rate over time
load balancing	avoid imbalances	traffic rate across IPs
traffic classification	correctness	packets by type

Table 1: Example use cases of our approach.

Of course, the techniques we present do not support all possible anomaly detection algorithms. Our main goal is to show the feasibility of in-switch statistical analyses. We hope that providing evidence of such statistical primitives can fuel additional research into in-switch anomaly detection, and unlock new network systems' designs (e.g., see Figure 1). We further discuss limitations and future directions in Sec. 5.

2 STATISTICS IN P4

We envision that switches autonomously analyze distributions of values of interest (e.g., traffic volume every 100ms, SYNs per destination IP, number of TCP vs UDP vs QUIC packets, etc.). Table 1 shows the values of interest pertaining to different use cases. Suppose that at a time t, a switch has extracted N values of interest $X = \{x_1, x_2, \ldots, x_N\}$ from the received traffic. We aim to characterize X, and keep our characterization updated for every packet received after t.

Approach. We keep one counter per value $x_i \in X$: incoming packets update counters and statistical measures of X. We first focus on mean, variance and standard deviation.

To compute those measures, we cannot rely on prior online algorithms (e.g., [26]), because P4 does not support division and square root operations that are used in the definition of mean, variance and standard deviation. Given $X = \{x_1, \ldots, x_N\}$, the mean is indeed defined as $\bar{x} = \frac{\sum_{i=1} x_i}{N}$, the variance as $\sigma_X^2 = \mathrm{E}[X^2] - \mathrm{E}[X]$ where $\mathrm{E}[f(X)] = \frac{\sum_{i=1}^{N} f(x_i)}{N}$ for any function $f(.)$, and the standard deviation σ_X as the

square root of the variance of X. We also avoid storing approximations in match-action tables (e.g., as done in [14]) because they require significant memory to be accurate.

For any distribution X of N values, we instead track $NX = \{Nx_1, \ldots, Nx_N\}$. In doing so, we leverage the insight that network anomaly detection often relies on *comparing relative values* of statistical measures. For example, if we want to check that the average traffic rate matches a value T, we can track packets per time interval as NX, and compare the mean $\overline{NX}$ of NX with $N \times T$. Also, if traffic rates follow a normal distribution, we can check if the rate x_j at any time j is an outlier by testing if $Nx_j > \overline{Nx} + 2\sigma_{NX}$.

Keeping one counter per value and tracking NX may seem excessive from a memory perspective. We note, however, that all the use cases in Table 1 require tracking distributions that inherently have a limited number of possible values (i.e., a low N). For example, monitoring traffic rates over the most recent time intervals (e.g., a few tens or hundreds of them) enables the detection of most failures and attacks. In many practical scenarios, we can further reduce memory consumption by storing the *order of magnitude* of the values in the tracked distributions, possibly relative to a *baseline*. For example, if we keep 100ms-long counters and a switch forwards 10Gb of traffic in most of the 100ms intervals, we can track values in Gb units: a counter with value 10 would then represent that the switch forwarded 10 Gb in the corresponding 100ms, and anomaly detection would not require to store values much bigger than 100. Similarly, to detect failures, we can keep the order of magnitude of stalled flows, as this likely changes when a failure occurs. We acknowledge that even the above approach has limitations. For example, it may be impractical to monitor all the possible values of a 64-bit header field. The investigation of use cases or applications requiring this sort of monitoring is left for future work.

Revisited definitions. Given $NX = \{Nx_1, \ldots, Nx_N\}$, we define X_{sum} and X_{sumsq} as follows:

$$X_{\text{sum}} = \sum_{i=1}^{N} x_i; \qquad X_{\text{sumsq}} = \sum_{i=1}^{N} x_i^2$$

By definition, the mean of NX is exactly X_{sum}. Additionally, for the variance σ^2_{NX}, we have:

$$\begin{aligned}\sigma^2_{NX} &= \mathrm{E}[(NX)^2] - \mathrm{E}[NX]^2 \\ &= \frac{\sum_{i=1}^{N} (Nx_i)^2}{N} - \left(\frac{\sum_{i=1}^{N} Nx_i}{N}\right)^2 \\ &= N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2 = NX_{sumsq} - X^2_{sum}\end{aligned}$$

The standard deviation σ_{NX} remains equal to $\sqrt{\sigma^2_{NX}}$.

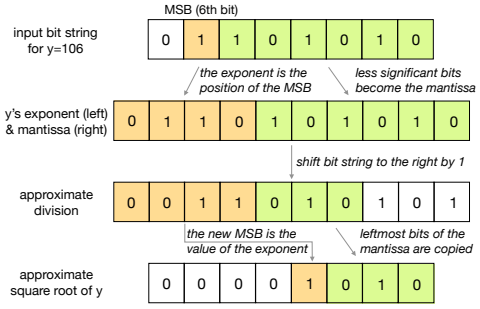


Figure 2: Example of our approximated square root algorithm: it approximates $\sqrt{106}$ to 10.

input number y	50th perc	90th perc	max
1-10*	3%	10%	20%
10-100	0.4%	1.4%	3.8%
100-1000	<0.05%	0.14%	0.44%
1000-10000	<0.01%	<0.01%	0.05%

* for small numbers, the percentage error is high but the absolute error is low – e.g., $\sqrt{3}$ approximated to 1

Table 2: Percentage error in square root estimation with respect to the fractional square root value.

Online computation of mean, variance and standard deviation. We instruct switches to update their counters and the values of N, X_{sum}, X_{sumsq} and σ^2_{NX} according to the received traffic. When we receive a new value of interest x_k (e.g., the traffic rate in a new 100ms sample), we increase N by 1, and X_{sum} by x_k. We also modify the value of X_{sumsq} by adding the square of x_k, and store x_k in a new counter.

We can also monitor *frequency distributions*, where each $x_i \in X$ represents the frequency of a value of interest (e.g., SYN vs data packets). In this case, when we receive a new value k, we increase N only if x_k is equal to 0. Before incrementing x_k by 1, we also increase X_{sum} by 1, and update X_{sumsq} by adding $(x_k + 1)^2$ and subtracting its old value x_k^2:

$$X_{\text{sumsq}} \leftarrow X_{\text{sumsq}} + (x_k + 1)^2 - x_k^2 = X_{\text{sumsq}} + 2x_k + 1$$

For both frequency and non-frequency distributions, we then recompute σ^2_{NX} as $NX_{sumsq} - X^2_{sum}$. Finally, to compute σ_{NX}, we approximate the square root of the variance with the following algorithm, exemplified in Figure 2.

Given the bit string b representing an integer y, we first compute y's floating point representation f: we set f's exponent to the position of b's most significant bit (MSB), and copy the bits after the MSB as f's mantissa. We then shift f by one to the right, obtaining a bit string y_1. Finally, we translate back y_1 into its integer representation b_1, setting b_1's most significant bit to the value of y_1's exponent, and copying y_1's mantissa into b_1's least significant bits.

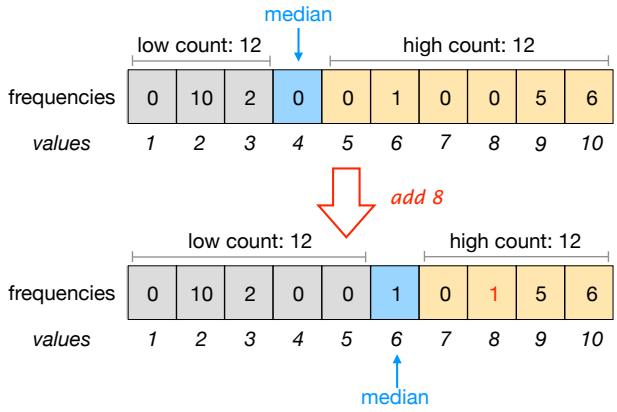


Figure 3: Example of our approximated algorithm to compute the median on a frequency distribution.

N	example use case	before $N/2$ samples 50%tile	before $N/2$ samples 90%tile	after $N/2$ samples 50%tile	after $N/2$ samples 90%tile
100	packet types	4.5%	34.5%	0%	1%
1000	per-ms traffic	3.6%	29.6%	0%	0.1%
65536	16-bit field	<1%	23%	0%	0.01%

Table 3: Median estimation error for distributions of N elements, over 20 repetitions per value of N.

Intuitively, the shifting operation divides the exponent by two, ensuring that the MSB of the computed square root is correct. It also divides the mantissa by two, which is indeed an approximation of the actual value of the square root. As such, the algorithm essentially computes an interpolation between subsequent squares of the form 2^{2k}. Table 2 shows the accuracy of this algorithm as reported in our experiments.

We note that some hardware switches do not support the squaring of values unknown at compile time. Similarly to our square root approximation, we can approximate squaring by using shifting operations, as also suggested in [7].

Online computation of percentiles. By monitoring frequency distributions, we can track values and change rates of percentiles, which may be indicative of anomalies.

Let's first consider the online computation of the median (i.e., the 50th percentile) of a distribution X. The median is defined as a value m such that half of the elements in X are lower than m and the other half are higher than m. To compute it online, we store the distribution $F = \{f_1, \ldots, f_N\}$ where each f_i is the frequency of x_i in X. We also track the combined frequency of all the values lower than the current median, and all the values higher than it, in two variables. This enables us to update the median for every new value x_j: if the combined frequency of values higher (resp., smaller) than the current median becomes bigger than the frequency of values lower (resp., higher) than the median plus the median itself, we move the median towards the higher (resp., lower) values. An example is shown in Figure 3.

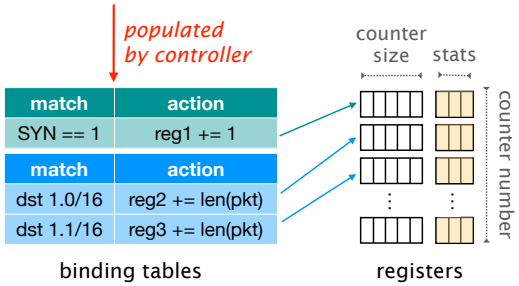


Figure 4: Switch resources for STAT4 computations.

A challenge of implementing this algorithm in P4 consists in skipping frequency counters with value equal to zero (e.g., 5 in Figure 3) when updating the median, because P4 does not support iteration. Since we want to avoid packet recirculation, our current approach is to move the median by at most one unit per packet: in Figure 3, it would therefore take us two packets to move the median from 4 to 6. This approach leads to an estimation error which is proportional to the size of F in the worst case – e.g., if the median moves from the beginning to the end of a very sparse distribution.

Yet, our estimation error tends to be low for non-sparse distributions. Table 3 shows the results of experiments where we feed our median computation algorithm with values extracted from a range $[1, \ldots, N]$. The estimation error is always $\leq$ 1%, except early in our simulations, when distributions are sparse. The error would be even lower when switches receive packets *not* carrying values of interest, as those packets *do* contribute to moving the median.

We support the online computation of any percentile by only adjusting the comparisons between the combined frequencies of values lower and higher than the current percentile. For example, tracking the 90-th percentile p amounts to ensuring that the frequency of values lower than p is nine times bigger than the frequency of values higher than p.

3 TOWARDS PRACTICAL IN-SWITCH ANOMALY DETECTION

We have implemented STAT4, a P4 library that `bmv2` programs can import to track distributions of values extracted from packets, as described in Sec. 2. As illustrated in Figure 4, STAT4 uses switches' registers to store the distributions and their statistical measures. The control plane decides which distributions to track at any time by populating P4 tables that we call *binding tables*. Main implementation details follow.

Lazy computation of standard deviation. Detection algorithms typically perform read operations on statistical measures much less frequently than updates occur. For example,

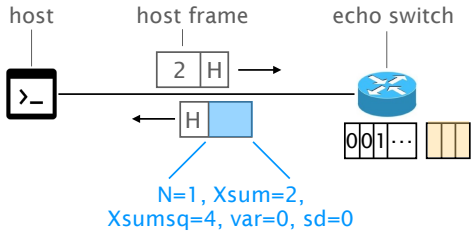


Figure 5: Experimental setup for STAT4 validation.

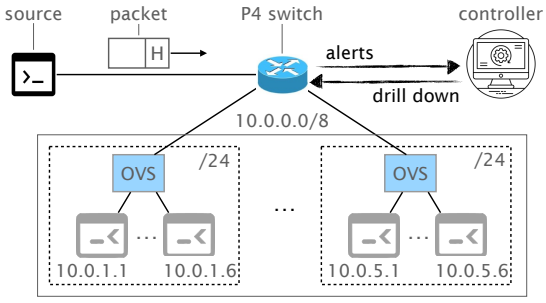


Figure 6: Experimental setup for our case study.

if we track traffic per second, every packet contributes to the amount of the traffic in the current second, but anomalies can only be detected at the end of each second. Following this observation, our library updates the statistical measures only when a new value is added to the corresponding distribution.

This lazy approach is especially relevant for standard deviation, since it amortizes the cost of identifying the most significant bit (MSB) in binary strings – required by our square root computation algorithm. For the library to be self-contained, STAT4 currently identifies MSBs using a sequence of `ifs`, which is a costly operation. An alternative for hardware implementations is to perform a longest prefix match on an ad-hoc TCAM table populated by the controller at switch startup.

Simultaneous tracking of multiple distributions. STAT4 stores each value of a monitored distribution in a distinct cell of dedicated registers. The size and number of those registers is controlled by two compiler macros whose values can be tuned by P4 applications using the library: the maximum number of distributions tracked simultaneously depends on the macro `STAT_COUNTER_NUM`, and the number of values per distribution on the macro `STAT_COUNTER_SIZE`.

Runtime tuning of values of interest. Monitoring systems generally need to track different distributions over time. Suppose that a switch monitors both the traffic rate and the number of SYN packets. During a SYN flood attack, acquiring more information on the targets of the attack may be much more important than monitoring the general traffic rate.

To support such scenarios within STAT4 applications, controllers can adjust at runtime the tracked distributions without recompiling the P4 application, by modifying the content of STAT4's binding tables. For each distribution, those tables' entries indeed define (i) how to extract values of interest from packets, and (ii) how to update which registers.

Validation We validate the correctness of our STAT4 implementation by building an echo application on top of it. For each packet it receives, this application instructs the switch to report the tracked statistical measures in a reply packet.

We simulate a minimal network with a single host connected to a `bmv2` switch running the echo application, as illustrated in Figure 5. The host sends Ethernet frames whose payload only contains a randomly generated integer between −255 and 255. The switch tracks the occurrences of the integers in the received frames. Every time the switch receives a packet, it therefore updates the frequency distribution of the integers' values, and replies with a frame including the updated statistical measures of the distribution. The host compares the values in every received packet with the corresponding statistical measures it computes in software.

In all our experiments (with up to 10,000 packets), the values of N, X_{sum}, X_{sumsq} and σ^2_{NX} stored at the switch are equal to those computed at the host, and the output of our online algorithms is consistent with results in Sec. 2.

4 CASE STUDY

We now demonstrate how in-switch statistical primitives can support prompt detection of anomalous traffic *and* identification of the recipient of such traffic. The latter is achieved by zooming in to progressively finer-grained traffic statistics.

Setup. We emulate the scenario shown in Figure 6: a network monitoring system aims to quickly detect traffic spikes for internal hosts called *destinations*, across which packets are supposed to be load-balanced. By default, we set 36 destinations in six /24 subnets of a /8 prefix. We abstract external hosts as a single traffic *source*. The monitoring system includes a P4 switch and a custom controller. The switch provides connectivity and runs statistical checks on the crossing traffic. The controller tunes such checks by managing entries in the binding tables of the switch.

Detection and drill down experiment. Initially, the switch only monitors the packets per time interval for the entire /8 prefix, continuously checking if in any interval, the rate is higher than the mean of the stored distribution plus two standard deviations. The switch implements a circular buffer that by default, stores 100 8ms-long time intervals.

After generating traffic uniformly across the destinations for a randomized time, the source starts sending much more traffic to a randomly selected destination. The switch is then supposed to detect the traffic spike, and alert the controller.

Our controller implements the logic to drill down into traffic spikes. Upon receiving a traffic-spike alert, it adds an entry to a binding table, requiring the switch to track the traffic per /24 subnet *in addition to* the packet rate for the /8 over time. The switch should now detect that one /24 subnet receives much more traffic than the others, and send a traffic-imbalance alert to the controller. In response to this second alert, the controller modifies the previously added entry so that the switch tracks the traffic per destination within the identified /24 *instead of* the traffic per subnet.

Results. We repeat the above experiment many times, with time intervals ranging from 8 ms to 2 seconds, and number of intervals between 10 and 100. In all the experiments, the switch detects the traffic spike *in the first interval* after the start of the spike. It also generates alerts as expected, and correctly identifies the destination of the traffic spike, which varies between simulation runs. Pinpointing the destination of each spike typically takes 2-3 seconds because of the interaction between the control and data planes.

Resource Consumption. The case-study application occupies 3.1KB. It entails at most one dependency between match-action rules [11], since at most two rules with independent actions match each packet. The longest dependency chain in our code has 12 sequential steps, used to override the oldest counter in distributions of traffic over time: this chain may be shortened by refining our implementation. While the mapping between the above dependencies and pipeline stages depends on both compilers and hardware targets, we expect that our code be deployable in most commercial targets, as they typically support more than 10 pipeline stages [22].

5 FUTURE DIRECTIONS

This work focuses on the feasibility of in-switch statistical analyses. Anomaly detection applications can already be built on the current Stat4 library, as exemplified in Sec. 4.

We acknowledge that our library code is not optimal. Among future improvements, we plan to support a more parsimonious use of memory. Stat4 currently allocates switch resources for every possible value in the tracked distributions, even if some values are never observed. We will explore techniques to avoid reserving memory for non-observed values (e.g., using hash-tables similarly to [23]) which would be especially beneficial for sparse distributions.

From a broader perspective, we hope that our work can inspire future research to fully support the architectural shift visualized in Figure 1, including the following directions.

Larger exploration of in-switch statistical primitives. This paper focuses on a few statistical measures that are useful for many distributions. It is not rare, though, that network systems have to deal with distributions that are not straightforward to characterize with the measures we currently support. For instance, the distribution of traffic per prefix may be zipfian [25].

In our approach, the controller has access to all the values of distributions tracked by switches, as they are stored in switches' registers. It can therefore learn about the distribution at runtime, and adapt the switch's anomaly detection approach accordingly. For example, if a distribution is bimodal, the controller can instruct switches to separately track and check the two modes of the distribution.

A full exploration of how to analyze a wider range of distributions, possibly performing statistical analyses across multiple switches, is an interesting direction for future work.

Combining in-switch and in-controller monitoring. In addition to supporting time-sensitive tasks, in-switch statistical analyses can unlock *new* network applications. For example, they could enable the data plane to reroute packets *before* congestion, when traffic *starts* to surge. Identifying such new applications and quantifying the gains obtained by a data-plane-only approach are promising research topics.

Obviously, in-switch computation has limitations too. For example, its scalability is a primary concern: switches may simply not have the resources required for some computations. In contrast, scalability is a strength of centralized architectures, as it is cheap to distribute and scale controllers.

We envision that future monitoring systems will profitably combine in-switch and controller-based techniques. For example, they may use in-switch anomaly detection to decide *when* a controller should extract sketches from switches, e.g., to properly process a received alert. We believe that the design, implementation, and evaluation of such systems should be a relevant item in our community's research agenda.

Extending data-plane programmability. We show that in-switch statistical checks are feasible despite the limitations of switches and P4. One natural question is whether future hardware can better support our approach.

Most challenges that we faced derive from *fundamental* performance constraints. For instance, supporting divisions and iteration in hardware would degrade switches' performance (e.g., delaying packet forwarding). We thus imagine that in the coming years, the elementary operations available in hardware will not change drastically, and we will still have to resort on approximations like those presented in Sec. 2.

Nevertheless, better support for network *applications* could be provided in future. For example, allowing data planes to autonomously modify match-action tables would avoid latency-expensive interactions with controllers, such as when drilling down into anomalies. Similarly, equipping switches with an extra parallel pipeline dedicated to control tasks, such as in-switch statistics, may relax constraints (e.g., for hardware stages) motivated by packet processing efficiency.

REFERENCES

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

[2] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *Proceedings of the 2013 ACM SIGCOMM Conference (SIGCOMM '13)*. Association for Computing Machinery, New York, NY, USA, 99–110.

[3] Stewart Bryant, Clarence Filsfils, Stefano Previdi, Mike Shand, and Ning So. 2015. Remote Loop-Free Alternate (LFA) Fast Reroute (FRR). RFC 7490. (2015). https://rfc-editor.org/rfc/rfc7490.txt

[4] Xiaoqi Chen, Shir Landau Feibish, Yaron Koral, Jennifer Rexford, Ori Rottenstreich, Steven A Monetti, and Tzuu-Yi Wang. 2019. Fine-Grained Queue Measurement in the Data Plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies (CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 15–29.

[5] Benoit Claise. 2004. Cisco Systems NetFlow Services Export Version 9. RFC 3954. (2004). https://rfc-editor.org/rfc/rfc3954.txt

[6] Cloudflare blog. 2020. Famous DDoS attacks. www.cloudflare.com/en-us/learning/ddos/famous-ddos-attacks/. (2020).

[7] Damu Ding, Marco Savi, and Domenico Siracusa. 2020. Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE Press, USA, 1–9.

[8] Mark Fedor, Martin L. Schoffstall, James R. Davin, and Jeff D. Case. 1990. Simple Network Management Protocol (SNMP). RFC 1157. (1990).

[9] Sam Gao, Mark Handley, and Stefano Vissicchio. 2021. Stat4 repository. https://gitlab.com/svissicchio/stat4. (2021).

[10] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. 2017. Dapper: Data Plane Performance Diagnosis of TCP. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. Association for Computing Machinery, New York, NY, USA, 61–74.

[11] Vladimir Gurevich. 2017. P4 mapping to Barefoot Tofino(tm). Talk at P4 Developers Day. (2017).

[12] T. Holterbach, E. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever. 2019. Blink: Fast Connectivity Recovery Entirely in the Data Plane. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation (NSDI'19)*. USENIX Association, USA, 161–176.

[13] 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 (CoNEXT '19)*. Association for Computing Machinery, New York, NY, USA, 285–291.

[14] Ralf Kundel, Jeremias Blendin, Tobias Viernickel, Boris Koldehofe, and Ralf Steinmetz. 2018. P4-CoDel: Active Queue Management in Programmable Data Planes. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, USA, 1–4.

[15] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A Better NetFlow for Data Centers. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI'16)*. USENIX Association, USA, 311–324.

[16] 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 (CoNEXT '16)*. Association for Computing Machinery, New York, NY, USA, 481–495.

[17] Stephane Litkowski, Ahmed Bashandy, Clarence Filsfils, Pierre Francois, Bruno Decraene, and Daniel Voyer. 2021. Topology Independent Fast Reroute using Segment Routing. (2021). Internet Draft.

[18] 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 (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 101–114.

[19] 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 2017 ACM SIGCOMM Conference (SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 85–98.

[20] P4.org Applications Working Group. 2020. In-band Network Telemetry (INT) Dataplane Specification v2.0. https://p4.org/specs/. (2020).

[21] Sonia Panchen, Neil McKee, and Peter Phaal. 2001. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176. (2001). https://rfc-editor.org/rfc/rfc3176.txt

[22] Naveen Kr. Sharma, Antoine Kaufmann, Thomas Anderson, Changhoon Kim, Arvind Krishnamurthy, Jacob Nelson, and Simon Peter. 2017. Evaluating the Power of Flexible Packet Processing for Network Resource Allocation. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, USA, 67–82.

[23] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *Proceedings of the Symposium on SDN Research (SOSR '17)*. Association for Computing Machinery, New York, NY, USA, 164–176.

[24] Daniel Ting. 2018. Data Sketches for Disaggregated Subset Sum and Frequent Item Estimation. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1129–1140.

[25] Jörg Wallerich, Holger Dreger, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. 2005. A Methodology for Studying Persistency Aspects of Internet Flows. *SIGCOMM Comput. Commun. Rev.* 35, 2 (April 2005), 23–36.

[26] B. P. Welford. 1962. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics* 4, 3 (1962), 419–420.

[27] Zhaoqi Xiong and Noa Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19)*. Association for Computing Machinery, New York, NY, USA, 25–33.

[28] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (NSDI'13)*. USENIX Association, USA, 29–42.

[29] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, Pengcheng Zhang, Dennis Cai, Ming Zhang, and Mingwei Xu. 2020. Flow Event Telemetry on Programmable Data Plane. In *Proceedings of the 2020 ACM SIGCOMM Conference (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 76–89.