Designing Traffic Monitoring Systems for Self-Driving Networks

Chris Misa
University of Oregon
cmisa@cs.uoregon.edu

ABSTRACT

Traffic monitoring is a critical component of self-driving networks. In particular, any system that seeks to automatically manage a network's operation must first be equipped with insights about traffic currently flowing through the network. Typically, dedicated traffic monitoring systems deliver such insights in the form of traffic features to high-level human or automated decision makers. Inspired by the exciting capabilities of programmable dataplanes and the persistent challenges of network management, the research community has focused on improving the flexibility and efficiency of traffic monitoring systems for a variety of management tasks. However, a significant gap remains between the traffic monitoring requirements of practical, deployable self-driving networks and the capabilities of current state-of-the-art systems. This short paper provides a brief background of traffic monitoring systems, discusses how their claims and limitations relate to requirements of self-driving networks, and proposes several open challenges as exciting starting points for future research. Addressing these challenges requires large-scale efforts in traffic monitoring techniques and self-driving network design, as well as enhanced dialog between researchers in both domains.

1 Background & Motivation

1.1 Traffic Monitoring Systems

Traffic monitoring refers to the process of observing *packets* flowing through the network and computing *metrics* for a particular goal. As shown in Figure 1, this involves the network data plane where packets are observed, computation of the desired traffic metrics (typically involving filtering and aggregation), and finally the "self-driving" automation system where the traffic metrics are used to make decisions about how to update network forwarding behavior. For example, a self-driving network controller might seek to observe DNS packets, compute total volume of DNS traffic to particular destinations, then deploy mitigation if traffic exceeds a volume associated with DDoS attacks [9].

The primary challenge in monitoring network traffic is dealing with high traffic volumes (*e.g.*, a single switch can process up to several Tbps). In order to deal with this challenge, modern traffic monitoring systems leverage hardware and/or software processing platforms at several points in the network as shown in Figure 2. For example, DNS packets

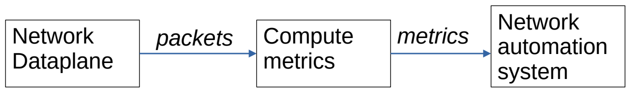

Figure 1: Traffic monitoring involves observing packets in the network and computing metrics for automation systems.

could be selected using TCAM-based match action tables in programmable switch hardware [4] or using logic implemented in CPU-based virtual switches [12].

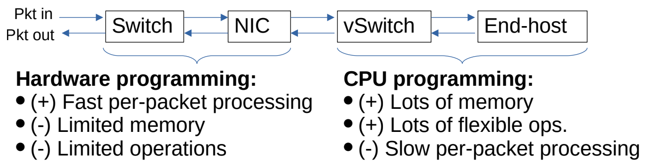

Figure 2: Processing platforms commonly considered in network traffic monitoring systems.

Although hardware processors (programmable switches and NICs) can efficiently process high traffic volumes, they do so by adopting simpler, constrained programming models with a limited set of per-packet operations (*e.g.*, limited read-update-writes per packet) and a small amount of memory (*e.g.*, $O(10\text{MB})$ SRAM on typical programmable switches). As a result, state-of-the-art traffic monitoring systems develop hybrid approaches where as much of the monitoring computation as possible is offloaded to high-efficiency hardware processors (*e.g.*, switches) while the rest is implemented in lower-efficiency CPU-base software. For example, Sonata [6] develops algorithms for partitioning monitoring computations across switch hardware dataplanes and CPU-based stream processors.

1.2 Self-Driving Examples

To illustrate how traffic monitoring relates to self-driving network control systems, we consider two recently proposed network automation systems.

DDoS defense. Recent proposals [9] develop approaches to automatically defending against network-based DDoS attacks by combining the data and control plan components described in Figure 2. The core idea is to install traffic monitoring programs directly into programmable switch hardware, then automatically react based on the collected metrics to mitigate attack traffic. Although presented as end-to-end defense systems, these proposals each leverage generic traffic monitoring capabilities which could be satisfied by a single unified monitoring system.

Flow-level offloading. Other proposals [14] seek to improve performance of modern cloud gateway routers by offloading processing (*e.g.*, encap-decap, forwarding) to hardware processors with limited memory. The core idea is to automatically select a few "heavy" flows for offloading to the hardware processor (*e.g.*, switch, NIC) so that the CPU handles reduced traffic volume. Again, the traffic monitoring requirements for self-driving flow offloading are generic (finding the "heaviest" and the "lightest" flows) and could be implemented by a unified system.

1.3 General Requirements

Based on these examples, we argue that unified traffic monitoring systems must meet the follow requirements to support current and future self-driving networks.

R1: Set of monitored metrics changes at runtime. Traffic monitoring systems must be able to change what metrics are computed at runtime on-the-fly. For example, the flow-offloading controller might need to adjust which offloaded flows to monitor or the DDoS defense controller might need to monitor new per-source metrics after detecting an attack (*e.g.*, to identify attack sources).

R2: Must retain resource efficiency for all metrics. Traffic monitoring systems must be able to maintain consistent accuracy for all metrics computed. For example, the flow-offloading controller might be able to achieve high performance even when the set of "heavy" flows reported from the monitoring system is computed approximately using a smaller amount of memory.

R3: Must remain robust in the face of changing traffic. Traffic monitoring systems must be able to cope with changes in resource requirements induced by the natural changes in traffic composition over time. For example, per-source metrics required by the DDoS defense controller might require memory proportional to the actual number of sources observed which changes dynamically over time.

2 Current Traffic Monitoring Design Patterns

Current state-of-the-art traffic monitoring system proposals focus primarily on addressing **R2**. We consider two key trends in this area: approximation using *sketches* and monitoring task definition using *query languages*.

2.1 Sketches for Efficient Approximation

Sketch-based methods [15] extend the core idea of a hash table to an approximation method for computing a keyed sum (*i.e.*, the number of packets or bytes in each flow). A "sketch" is essentially a hash table which embraces hash collisions—rather than implementing collision resolution, a sketch adds multiple semi-independent hash functions. As more hash functions are added, the probability of hash collision (*i.e.*, all hash functions hashing two different elements to the same buckets) decreases multiplicatively so that when properly parameterized and under a few other assumptions, the error induced by hash collisions can be provably bounded.

The key advantages of sketch-based methods is that their update algorithm is constant time ($O(1)$) and that they can estimate several useful metrics beyond simple keyed sums. Hash-indexed read, increment, write operations are relatively trivial to implement on modern programmable switch hardware making sketches an easy first choice for nearly all switch hardware based traffic monitoring proposals. Moreover, in addition to simple per-flow counting, metrics like heavy hitters, cardinality, and entropy can also be estimated from sketch counters [8].

Despite their promise and popularity, several key limitations have hindered the practical application and adoption of sketch-based methods in realistic traffic monitoring settings. First, sketches typically require fixing a flow key at compile time making it challenging to address **R1** since either sketches for all possible metrics must be run all the time or the monitoring program must be recompiled and redeployed (inducing network down time). Several recent works [7, 17] tackle this challenge head on, but the effectiveness of the proposed methods remains untested for self-driving network applications. Second, the accuracy of sketch-based results is strongly dependent on the relationship between the number of counters compiled in the sketch (*i.e.*, the number of rows in the "hash table") and the actual number of flows observed in network traffic. This inherently limits a sketch's ability to address **R3** since the actual number of flows that must be tracked in realistic network traffic can change drastically over time and it is nearly impossible to select an optimal number of sketch counters a priori.

2.2 Query Languages for Flexibility

Another focus of recent traffic monitoring research is in developing expressive languages for expressing monitoring tasks (often referred to as "queries") which can be automatically compiled into high-throughput platforms like programmable switch hardware. In particular, a form of map-reduce language has emerged as a promising design choice since it enables complex processing pipelines and has a relatively straightforward mapping into the "match-action" model of modern programmable switch hardware [11, 6].

The key advantage of developing a unified language for expressing traffic monitoring computations is that it separates developers of self-driving control systems from the technical low-level interfaces (*e.g.*, P4 [3]) where these computations are implemented. For example, the set of benchmark queries originally proposed in Sonata has been used to demonstrate performance of several other traffic monitoring systems [19, 10] implying that a self-driving network that uses queries in the Sonata language could be ported across multiple traffic monitoring "backends". Moreover, recent developments [19, 18] have demonstrated how such a language can be mapped to a more flexible hardware "interpreter" so that queries can be changed on-the-fly satisfying **R1**.

Despite the success of these initial efforts, current query languages are still limited in the types of aggregations they can express (*i.e.*, **R2**) as well as their robustness against changing traffic compositions (*i.e.*, **R3**). First, aggregation operations are typically selected from a list of pre-defined options and typically only support a few options like "sum", "count", and "average". In particular, specifying aggregations in this manner makes it challenging to implement more complex pattern-based queries (*e.g.*, as proposed in NetQRE [16]). Finally, similar to sketch-based methods, each aggregation expressed in these map-reduce languages must be mapped to a fixed-size hardware table whereas the actual number of aggregates (*i.e.*, number of observed keys) changes dynamically at runtime. For works like Newton [19] which propose using sketches to implement aggregations, the implications of error propagation through the query's pipeline of operators is unclear and potentially renders final query results useless.

3 Open Research Challenges

Finally, we summarize two key open research challenges implied by the limitations of prior traffic monitoring systems and the unique requirements of self-driving networks.

3.1 Role of Traffic Monitoring

As traffic monitoring systems develop new capabilities and complexities, a key question of where to draw the line between monitoring and control arises. Consider, the use of machine-learning (ML) models as a means to automatically make network control decisions [5, 13, 1]. Without a clear definition of the role of traffic monitoring, self-driving network efforts risk either over-looking key technical challenges required to collect features for these models efficiently at scale or risk duplicating efforts from traffic monitoring research.

3.2 Dynamic Resource Management

To satisfy both **R1** and **R3**, self-driving networks require that traffic monitoring systems produce consistently accurate results as both the metrics monitored as well as the traffic composition (*e.g.*, number of flows) changes dynamically over time. Although previous works address these requirements in isolation [19, 10, 2], addressing both requirements simultaneously for the wide range of metrics required remains an open challenge.

4 Conclusion

The brief overview presented here illustrates how network traffic monitoring is a rich field with a variety of challenging requirements and open problems as well as its essential role in the design and implementation of self-driving networks. Collaboration between traffic monitoring and automated control systems research will be critical for development of useful, practical, and effective self-driving networks of the future.

5 References

[1] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *NDSS*, 2021.

[2] R. Bhatia, A. Gupta, R. Harrison, D. Lokshtanov, and W. Willinger. Dynamiq: Planning for dynamics in network streaming analytics systems. *arXiv preprint arXiv:2106.05420*, 2021.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[4] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[5] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del Rincon, and D. Siracusa. Lucid: A practical, lightweight deep learning solution for ddos attack detection. *IEEE Transactions on Network and Service Management*, 17(2):876–889, 2020.

[6] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[7] Q. Huang, P. P. Lee, and Y. Bao. 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)*, pages 576–590, 2018.

[8] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114. ACM, 2016.

[9] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar. 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)*, 2021.

[10] C. Misa, W. O'Connor, R. Durairajan, R. Rejaie, and W. Willinger. Dynamic scheduling of approximate telemetry queries. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 701–717, 2022.

[11] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 85–98, 2017.

[12] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, et al. The design and implementation of open vSwitch. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, pages 117–130, 2015.

[13] T. Swamy, A. Zulfiqar, L. Nardi, M. Shahbaz, and K. Olukotun. Homunculus: Auto-generating efficient data-plane ml pipelines for datacenter networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 329–342, 2023.

[14] Y. Wang, D. Li, Y. Lu, J. Wu, H. Shao, and Y. Wang. 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)*, pages 535–550, 2022.

[15] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with OpenSketch. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 29–42, 2013.

[16] Y. Yuan, D. Lin, A. Mishra, S. Marwaha, R. Alur, and B. T. Loo. Quantitative network monitoring with netqre. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 99–112, 2017.

[17] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang. Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 207–222, 2021.

[18] H. Zheng, C. Tian, T. Yang, H. Lin, C. Liu, Z. Zhang, W. Dou, and G. Chen. Flymon: enabling on-the-fly task reconfiguration for network measurement. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 486–502, 2022.

[19] Y. Zhou, D. Zhang, K. Gao, C. Sun, J. Cao, Y. Wang, M. Xu, and J. Wu. Newton: Intent-driven network traffic monitoring. In *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 295–308, 2020.