Revisiting Network Telemetry in COIN: A Case for Runtime Programmability

Chris Misa, Ramakrishnan Durairajan, Reza Rejaie (University of Oregon), and Walter Willinger (NIKSUN, Inc.)

***Abstract*— Applications based on the Compute-In-the-Network (COIN) paradigm require flexible network telemetry data to drive effective allocation decisions. Telemetry systems collect such data based on queries specifying the precise traffic metrics or features required. Recent advances in programmable switch hardware have led to highly efficient methods to compute query results using in-network resources. However, current approaches fail to meet the simultaneous requirements of dynamic traffic loads, diverse query types, and query dynamics.**

In this work, we argue that telemetry systems should be cast as active runtime schedulers rather than static data sources. COIN-based applications can then dynamically submit telemetry queries on-the-fly and react to their results in a closed-loop fashion—enabling a new generation of telemetry-driven reactive applications. As a first step towards this vision, we present a single-switch telemetry operation scheduling method. Our empirical evaluation demonstrates that such a method can reduce overheads by an order of magnitude compared to worst case allocations. This initial exploration opens the door to a multi-tier scheduling framework combining switch hardware with software-based compute platforms to meet the telemetry demands of future COIN-based applications.

I. Introduction

Network telemetry data is critical for realizing the Compute-In-the-Network (COIN) paradigm and driving management, load-balancing, and security decisions for edge applications. To illustrate, consider a video streaming platform that transmits live video streams between many users at the edge while simultaneously converting streams to a number of different formats and resolutions on-the-fly using cloud and edge compute resources [1]. To reap the promised benefits of a COIN-based realization of such a platform, allocation decisions must be made based on detailed telemetry data about the number of users and volumes of video traffic at each node in the edge-cloud continuum. While application-level logs could be used in the context of specific software implementations, traffic-level telemetry data yields complete and application-agnostic metrics to drive dynamic allocation decisions in such deployments.

Specifically, network telemetry systems fulfill a critical requirement of COIN-based applications by providing visibility into current traffic conditions. Operators submit high-level, precise, and generic queries over all packets flowing through the network and the telemetry system returns query results, similar to querying a streaming database. For example, a telemetry system could allow operators of the distributed video streaming platform to query the network to determine how many clients are actively streaming from each content delivery node at a given point in time. Suppose a particular content delivery node is overloaded with clients. Operators could then submit follow-up queries to profile the distribution of connected clients and use this information to drive intelligent reallocation decisions (*e.g.*, by deploying a new CDN node in a location central to the currently observed clients). This example hints at several key requirements imposed by COIN-based applications on network telemetry systems. In particular, telemetry systems must be able to *(i)* efficiently compute high-level traffic statistics (such as the number of distinct hosts observed), *(ii)* provide wide visibility into traffic at different points in the network (*e.g.*, edge-to-fog traffic as well as fog-to-cloud traffic), and *(iii)* dynamically react to changing traffic compositions (*e.g.*, the number of active clients) and changing query workloads (*e.g.*, supporting arbitrary follow-up queries to inform (re)allocation decisions).

Since telemetry systems rely on in-network components to gather and process data, they are themselves COIN-based services which can be implemented and deployed along side the applications they serve. Prior efforts [2] demonstrated that programmable switch hardware allows telemetry systems to execute parts of the query processing operations directly in the data plane, reducing the volume of data that must be exported by several orders of magnitude. The main idea is to partition the query's data processing operations between switch hardware and a software-based stream processing platform. Intermediate query results are exported from the switch hardware to the stream processor which forwards the final results to human operators or automated COIN application managers. Other efforts have developed novel approximation techniques for particular query types (*e.g.*, sketches [3]) and investigated the possibility of changing query operations on-the-fly [4].

Despite the recent advances of in-network telemetry system design, a fundamental question remains: *how should such new switch hardware capabilities be incorporated into a complete end-to-end system for answering arbitrary dynamic queries over realistic traffic as required by edge applications?* Until very recently, switch-based telemetry system design followed a static approach, assuming that changes to running queries or traffic compositions would require full reloading of switch hardware programs (a process incurring seconds of network downtime [4]). On the other hand, a number of works have addressed the challenges of scheduling data processing graphs, a generic version of the particular processing operations required by telemetry queries, on heterogeneous software-based systems [5].

In this work, we present the next critical step towards realizing the envisioned telemetry systems for COIN-based applications by developing techniques for dynamically scheduling telemetry operations on switch hardware. After reviewing state-of-the-art approaches (§II), we describe a novel time-division scheduling method (§III), present empirical evidence supporting our proposed techniques (§IV), and conclude with a discussion of open challenges and opportunities (§V).

II. Background & Motivation

A. End-to-end Telemetry Systems

To illustrate the context, requirements, and challenges of dynamically scheduling telemetry operations, we consider a simplistic telemetry system, as shown in Figure 1, based on a network probe implemented on a single, centrally located enterprise or campus switch. This probe is controlled by a central entity which we refer to as the collector, implemented either on enterprise-local servers or in the cloud. Network operators or automated COIN application managers submit telemetry tasks to the collector which then computes a schedule for the required telemetry operations and configures the switch and other processing platforms (local cluster, remote cloud) to execute this schedule. As the schedule executes, the results of telemetry queries are returned to the collector for operator visualization, archival storage, or automatic forwarding plane updates. Well-known protocols are established for all of these communication channels as noted in the figure. We explain each of these components below.

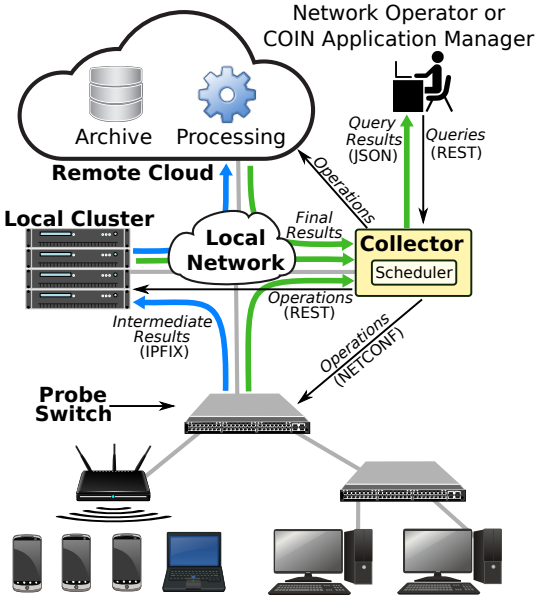


Fig. 1: Example network telemetry system architecture.

Diverse Telemetry Queries. Following Sonata [2], network operators express queries as directed acyclic graphs (DAGs) of atomic operations which progressively refine a stream of keyed-tuples from per-packet features (*e.g.*, header values) to high-level results (*e.g.*, the number of distinct sources). A limited set of such atomic operations (*e.g.*, filter, map, reduce) can describe a wide range of queries. For example, a query might filter for packets to or from the current set of video content delivery nodes, group these packets by unique TCP connections, and count the total number of such connections maintained by each content delivery node. Given massive traffic volumes, limited compute power, and the fact that in many scenarios exactness of telemetry results matters less than efficiency, techniques like sampling and sketching [3] are often used to produce approximate results with reduced resource requirements. For example, if the result of the connection-counting query mentioned above is used to decide when to launch a new content delivery node, computing the precise number of connections matters less than identifying when the number of connections is *large enough* to warrant a new allocation.

Processing Platforms. To execute the atomic operations described by a query, telemetry systems draw on three main processing platforms: *(i)* switch hardware, *(ii)* a local stream processing cluster, and *(iii)* a remote cloud service. First, fueled by recent advances in programmability, switch hardware [6] offers a high-performance platform to execute operations directly in the data plane at line rate. However, switches have a severely limited processing capacity and cannot execute some of the required operations for telemetry tasks such as floating point division. Second, stream processors can perform a larger set of general operations, such as the floating point linear algebra required for many ML techniques. However, moving raw data from the switch to the stream processor is challenging due to the increasingly high rates of network traffic which also require enormous computing power. Such computing power may not be feasible for an organization to maintain locally. Finally, cloud platforms are able to provide nearly limitless compute power on-demand while mitigating the overheads of maintaining a large pool of local resources. However, moving data to the cloud incurs latency, may be subject to bandwidth constraints, and further raises privacy and security concerns. Note that we do not consider end hosts as processing platforms because instrumenting end hosts is impractical in many network settings (*e.g.*, enterprise or campus networks).

System-Wide Scheduling Problem. Given a set of queries to execute and their associated atomic operations, a telemetry system must determine how best to execute these operations on the given processing platforms. Since the set of operations and their resource requirements may change over time in response to query and traffic dynamics, the allocation of operations to processing platforms must be periodically updated. To simplify matters, we assume that the telemetry system operates over fixed-duration time windows, called *epochs*, and only changes allocations at epoch boundaries. The scheduling problem is then as follows: *for each epoch, given the current operations, their resource requirements, and the available processing platforms, determine what platform should execute which operation.* We note that this system-wide definition of the scheduling problem must also be broken down into sub-problems for each platform. In particular, the switch platform must determine which hardware stage should execute which operation while the local cluster and remote cloud must assign operations to compute nodes.

Switch Hardware Sub-Problem. Due to its monitoring vantage point on the network forwarding path and its line rate

processing speed, we argue that switch hardware is an essential ingredient in the design of telemetry systems. However, as noted in §I, how best to schedule telemetry operations on switch hardware given this platform's particular constraints is not well understood. This work develops an initial approach to the switch hardware scheduling sub-problem, laying the ground work for integrating switch hardware into the system-wide scheduling problem stated above.

B. Key Challenges in Scheduling for Switch Hardware

We consider the following four challenges as critical components of the switch hardware scheduling sub-problem in the development of network telemetry systems for COIN-based applications.

Challenge 1 (C1): Variable Traffic Loads. Resource requirements of telemetry queries often depend directly on properties of current network traffic, such as the number of flows, leading to changes in resources required over a query's execution time. For example, consider a query which monitors the number of bytes exchanged with each client of a CDN node. This query must maintain a counter for each active client and so the total number of counters required is equal to the number of active clients, which is likely to change. Sketch-based estimation techniques are also impacted due to the fact that the traffic distribution determines the provable accuracy bounds. If resources are statically assigned to task operations, the telemetry system must conservatively allocate for worst-case scenarios leading to inefficient utilization of switch hardware resources. *To maximize the utilization of switch resources, telemetry systems should adjust resource allocations at runtime based on the observed traffic [7].*

Challenge 2 (C2): Query Diversity. The potential diversity of telemetry queries described in §II-A implies a wide range of different data aggregation granularities, latency requirements, and accuracy tradeoffs. For example, an operator might want to simultaneously track source, destination pairs experiencing high TCP re-transmission rates as an indicator of congestion while obtaining an overall estimate of source entropy to detect changes in client to CDN node relationships. The first query requires operations that keep track of packet sequences in each TCP flow with high accuracy though the intended use of this query's results imply that some latency is acceptable. The second query requires operations that summarize a property over all packets on the network allowing for some trade-off of accuracy while maintaining stringent latency requirements. *Telemetry systems should allow flexible query specifications, such as the stream processing model, as well as specification of per-query accuracy and latency goals.*

Challenge 3 (C3): Query Dynamics. Given the large number of management issues and security threats that need to be dealt with, telemetry systems may be called upon to evaluate a large number of tasks concurrently (*e.g.*, thousands of tasks [7]) and this number may vary significantly over time. Suppose a telemetry system is servicing a large number of COIN-based applications like the video streaming application described in §I. During normal operation, each of these applications may only require a single, simple query to monitor their base-line activity. However, in the worse-case, all of these applications could detect allocation imbalances and begin submitting follow-up queries at the same time, leading to a spike in the number of query requests submitted to the telemetry system. Ultimately, *telemetry systems should maximize the number of queries that can be executed given arbitrary query arrival patterns and the available resources while meeting the specified accuracy and latency goals.*

Challenge 4 (C4): Network-Wide Execution. The traffic which a query applies to may be split among multiple sections of the network or may pass through multiple devices eligible for executing telemetry operations. In such cases, the query's operations must be carefully distributed across the network to ensure that all relevant traffic is observed. The current work *does not* address this challenge, noting that a solid understanding of the challenges of scheduling telemetry operations on a single switch is prerequisite to understanding the network-wide challenge. In future work, we plan to investigate the design of a network-wide scheduler which generates a global schedule based on topology and forwarding information that is then translated to multiple independent switch-level schedules.

C. Prior Efforts and Their Limitations

Though prior efforts have developed telemetry systems that leverage switch hardware to satisfy a number of the challenges discussed in §II-B, no single approach is capable of simultaneously addressing all challenges posed by an end-to-end deployment for COIN-based applications.

Hardware-Only Approach. A number of efforts show that telemetry queries can be implemented using switch hardware [2], but these solutions focus on one particular type of task and do not consider the challenges of simultaneously adapting telemetry operations to traffic and query dynamics.

Hardware-Software Hybrid Approach. A recent innovative approach to switch-based telemetry system design, taken in Sonata [2], is to combine switch hardware with a software stream processing system. This approach satisfies C2 by extending beyond limited hardware. However, allocation decisions are made statically so this approach struggles to satisfy C1 and C3.

Dynamic Allocation Approach. Another approach, taken by DREAM [7], is to dynamically adjust the allocation of telemetry resources to tasks on-the-fly based on estimated task accuracy. While this approach addresses C1 by making adaptive allocations, it requires specialized accuracy estimation methods limiting its ability to address C2.

In-Band Approach. Several prior works have explored the possibility of applying adaptive sampling methods to in-band network telemetry (e.g., Sel-INT [8]). However, these methods focus on monitoring the state and behavior of network devices and do not consider the problem of scheduling multiple telemetry queries on limited switch hardware resources.

III. Scheduling Switch Resources

In this section, we outline design issues and argue that our approach satisfies C1 by dynamically updating resource

allocations, C2 by leveraging a telemetry interpreter built into switch hardware, and C3 by maintaining a dynamic pool of queries.

A. Example Scenario

To concretely illustrate the role of a switch resource scheduler, we return to the video streaming platform discussed in §I. Suppose that, at a certain point in the platform's deployment, two content delivery nodes are active and running on near-edge clusters in separate parts of the network. Now, suppose that at some point in time, a large number of new clients form connections to one of these nodes leading to a load imbalance. To detect this load imbalance, the platform must periodically submit queries to monitor the number of connections made to each CDN node. Once an imbalance is detected, the platform has a large number of potential actions to re-balance load along the edge-cloud continuum. While the current effort does not investigate the details of these actions, we note that any action should be informed and triggered by detailed traffic-level telemetry information describing features such as the distribution of bytes per flow, the source entropy among clients, or per-client packet-loss statistics. The scheduler for switch resources must be prepared to answer these types of time-varying queries from a potentially large number of COIN-based requesting applications.

B. Scope of Our Approach

For simplicity, in the current work we consider only interactions between the switch and collector (leaving the integration of a local cluster and cloud resources for future work) and assume (see §II-A) a central collector which coordinates the execution of telemetry tasks using a single switch's hardware resources over a series of epochs. At the beginning of each epoch, the collector decides which operations from the pool of currently submitted queries can and should be run on switch hardware and sends a command to the switch with instructions for how to execute these operations (*i.e.*, a hardware configuration). During the epoch, the switch hardware executes these operations over all packets flowing through the switch at line rate. At the end of the epoch, operation results in the form of aggregate counters are either returned to the collector or sent to the stream processor for further operations (as shown in Figure 1). This process repeats indefinitely during the telemetry system's deployment as submitted queries are answered and new queries are submitted.

C. Runtime Programmable Approach to Switch Hardware

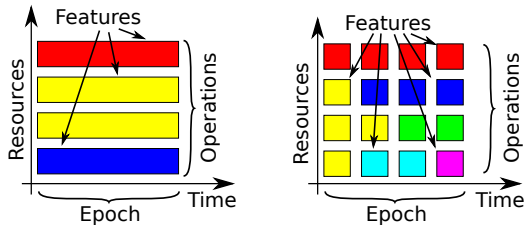
As COIN-based applications change their allocation decisions and telemetry requirements, the operations to be executed on switch hardware must also change on the fly, a capability that was considered infeasible until recently. Programmable switch hardware generally entails two stages of determining hardware operations. In the first stage, a static hardware description is written in languages like P4 [9] and compiled into a binary executable determining how generic hardware resources (*e.g.*, match-action tables, SRAM) are to be connected and configured. This static configuration exposes a control plane API which allows the second runtime stage to control some aspects of packet forwarding dynamically. Previous telemetry proposals compiled query operations into the static switch program, allowing great flexibility of query operations, but limiting the ability to change queries during runtime.

A key enabler of the runtime approach to switch programming is the observation that, by exposing a sufficiently rich parameter space, the static hardware program essentially acts as an *interpreter*, executing any telemetry operations configured via the control plane. Several recent efforts have leveraged this observation and described systems where queries can be added and removed from switch hardware on-the-fly [4]. We envision such telemetry operation interpreters can be statically compiled into the hardware pipelines of one or more switches in a network to provide the execution environment for runtime telemetry operations.

D. Switch Scheduling Sub-Problem

Given a set of queries and their associated DAGs of operations, as well as a runtime telemetry operation interpreter (§III-C) compiled into a static switch pipeline, the switch hardware scheduling sub-problem determines how best to assign resources (*e.g.*, ALU operations, memory) to operations in each epoch. We assume that local stream processor and cloud platforms are available to execute operations that cannot fit in switch hardware, but focus only on issues around scheduling for switch hardware. Given this focus, the main goal of the scheduler is to maximize switch utilization since doing so reduces load on other platforms.

Inputs to Scheduler. In addition to descriptions of what operations to run and the available switch resources, the scheduler must also take into account how much and what type of resource each operation requires. We determine these requirements based on history, maintained by the collector, of the past resource requirements for each operation. Historic resource requirements can be maintained automatically for any query specified as a generic DAG of operations by maintaining per-operation estimates.



(a) Original epoch. (b) Subdivided epoch.

Fig. 2: Dividing epochs with time-division multiplexing exposes more opportunities for scheduling operations on fixed switch hardware resources.

Time-Division Multiplexing. Runtime programmability enables time-division multiplexing of switch resources to trade-off slightly reduced accuracy for increased scalability. Specifically, available telemetry resources can be multiplexed across

different sets of operations over time, allowing the system to ultimately execute a larger number of tasks in pursuit of C3. Figure 2 shows an example of two resource multiplexing strategies. In Figure 2(a), resources are allocated to $n = 4$ operations during an epoch of length τ where n is limited by the available hardware resources. In Figure 2(b), the epoch duration is divided into $k = 4$ subepochs of length $\tau' = \frac{\tau}{k}$ and resources can now be allocated to operations across any combination of $k * n$ slots per epoch. In cases where an operation is not executed in each consecutive subepoch, cluster sampling theory provides estimates of the induced error *after gathering subepoch results* to drive future allocation decisions. This is in stark contrast to sketch-based approximation methods where assumptions about traffic composition are built into table sizes and not easily verifiable or modifyable based on observed results alone.

Optimal and Heuristic Solutions. Given the inputs described above, the task of the switch hardware scheduler can be mapped into a constrained optimization problem. At a high level, constraints are imposed by switch hardware resources and objectives are formulated in terms of maximizing operation accuracy or minimizing traffic sent to the collector. While prior efforts attempted to solve similar optimization problems statically based on large, representative traffic samples [2], we note that in the runtime programmable approach, schedules can be updated incrementally in response to observed traffic compositions. Our approach of maintaining aggregate moving averages of the traffic features relevant for scheduling decisions allows formulation of simpler optimization problems which can often be solved by modern optimization frameworks in tens of milliseconds. When an optimal solution cannot be found, our system falls back on heuristic approaches.

IV. Initial Evaluation

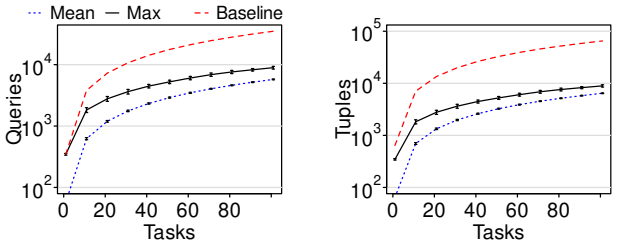
In this section, we offer initial empirical evaluation of the techniques described previously, showing that *(i)* dynamic allocation can reduce hardware resource requirements and *(ii)* our time-division approximation method yields comparable performance to other approximation methods while offering an efficient implementation path .

Hardware Prototype. We develop a prototype telemetry system based on a Broadcom switch ASIC with the BroadScan runtime programmable telemetry module. This module implements the telemetry operation interpreter described in §III-C and allows operations to be modified by efficient DMA-accelerated writes into ASIC memory. We evaluate our prototype by using `tcpreplay` [10] and replay the 2019 CAIDA Internet traces [11] from a server connected to the switch via a 40Gbps optical link.

Metrics. Since each counter must be exported as a result tuple, we use the number of tuples sent to the collector as a proxy for memory usage in our prototype. This metric is also an indicator of load on the collector or other down-stream processing platforms.

A. Dynamic Resource Allocation

First, we address the question of how dynamically scheduling shared resources between queries can improve resource requirements when scaling to large numbers of monitoring tasks. We start by considering a task based on the MRT algorithm [12] that submits a variable number of queries to iteratively zoom in on heavy-hitters in the CAIDA trace. On our prototype system, this task takes ∼23 1-second epochs to find the heavy hitters. For each epoch, we gather the number of queries submitted and the number of tuples returned yielding an excerpt of the task's resource requirements over time. To approximate overall resource requirements, we combine several such excerpts in an epoch-level simulation. In particular, we simulate the resource requirements of a given number of tasks executing over 1000 epochs where each task submits a particular empirically-measured resource excerpt in a loop separated by a gap determined by a Poisson distribution with a mean rate of 2 seconds.



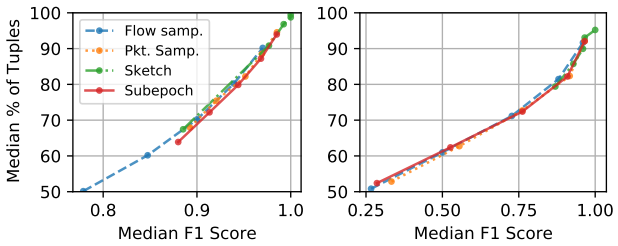
(a) Queries submitted. (b) Tuples returned.

Fig. 3: Per-epoch resource usage when running multiple tasks showing worst case and the observed maximum and mean values per epoch. Error bars show std. deviations over 100 trials of the simulation.

Figure 3 shows the mean and maximum of the per-epoch resource requirements computed in our simulation. We also plot baseline requirements based on a worst-case static allocation for all simulated tasks. These results illustrate that dynamic allocation can potentially reduce the resource requirements by an order of magnitude when multiple tasks are contending for shared resources—said differently, an effective solution to C3 improves overall system scalability.

B. Performance of Time-Division Multiplexing

Next, we evaluate the performance of our time-division multiplexing technique (§III) in its ability to reduce load on the collector while maintaining reasonable accuracy. We consider an example query from Sonata [2] which detects potential DDoS attack victims by selecting destinations receiving from more than a given number of sources. We measure query accuracy as the F1 score (harmonic mean of precision and recall) compared to ground truth and resource requirements as the number of tuples returned to the collector. In addition to our subepoch multiplexing scheme, we compare with three conventional approaches: flow sampling, which takes a simple random sample of all five-tuple flows [13]; packet sampling, which takes a simple random sample of all packets; and sketching which uses a bloom filter to keep track of distinct elements in the first query operation [4]. We run these experiments in simulation over two representative traces of ISP-level traffic from CAIDA [11] and MAWILab [14] (captured on Sept. 1st, 2019).



(a) CAIDA (b) MAWILab

Fig. 4: Comparison of accuracy vs. tuples returned for time-division multiplexing and conventional approaches.

Figure 4 shows the median percentage of tuples (compared to an oracle method) against the median F1 score for the four techniques considered. The key observation is that time-division multiplexing achieves a similar tradeoff between measurement accuracy and the number of tuples returned to the collector compared to conventional approaches. However, unlike conventional approaches, time-division multiplexing generates sound error estimations based on observations alone and is hence well-suited for dynamic telemetry systems.

V. Outlook

Our evaluation in §IV demonstrates that dynamically scheduling operations in switch hardware can improve the scalability of this platform with respect to traffic and query load. However, a number of critical challenges and opportunities arise when extending this approach to a network-wide deployment throughout the edge-cloud continuum. First, we note that telemetry systems can directly leverage techniques developed for other COIN applications, such as sensor networks [15], to efficiently schedule operations on edge and cloud software platforms. This opportunity raises several challenges in the need to preserve the privacy and anonymity of network traffic when incorporating compute entities outside of the original network. Second, while we focus on a scenario where the telemetry scheduler has direct access to centrally-located switch hardware, as hinted at in C4 (§II-B), COIN-based deployments may need to monitor traffic distributed across several sections of the network or services deployed across heterogeneous administrative domains. For example, a telemetry system might need to distribute query operations across programmable switches on the edge and generic telemetry data provided by cloud services in the core. Critical issues must be addressed in how to virtualize query operations over these diverse data sources. Finally, we note the need to address a multi-tiered scheduling problem to allocate query operations on each particular compute platform (switch hardware, edge stream processor, cloud resources) and to coordinate scheduling between different platforms. Scheduling solutions for each particular compute resource must ultimately be united in a global scheduling framework which maps query-level goals, such as accuracy or latency, into the specific switch, stream processor, and cloud platforms.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. We also thank Shahram Davari and Broadcom, Inc. for providing hardware and technical support for our prototype telemetry system. This work is supported by the National Science Foundation through CNS 1850297, a Ripple faculty fellowship, and a Ripple graduate fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, Ripple, or Broadcom.

References

[1] G. Gao and Y. Wen, "Video transcoding for adaptive bitrate streaming over edge-cloud continuum," *Digital Communications and Networks*, 2020. https://doi.org/10.1016/j.dcan.2020.12.006. Accessed: May 17th, 2021.

[2] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: query-driven streaming network telemetry," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 357–371, 2018.

[3] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation*, pp. 29–42, 2013.

[4] 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 16th International Conference on emerging Networking EXperiments and Technologies*, pp. 295–308, 2020.

[5] A. Shukla and Y. Simmhan, "Model-driven scheduling for distributed stream processing systems," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 98–114, 2018.

[6] "Product brief tofino page." https://barefootnetworks.com/products/brief-tofino/. Accessed: May 17th, 2021.

[7] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: dynamic resource allocation for software-defined measurement," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 419–430, 2014.

[8] S. Tang, D. Li, B. Niu, J. Peng, and Z. Zhu, "Sel-int: A runtime-programmable selective in-band network telemetry system," *IEEE transactions on network and service management*, vol. 17, no. 2, pp. 708–721, 2019.

[9] 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*, vol. 44, no. 3, pp. 87–95, 2014.

[10] "Tcpreplay - pcap editing and replaying utilities." https://tcpreplay.appneta.com/. Accessed: May 17th, 2021.

[11] "The CAIDA UCSD anonymized Internet traces dataset - 2019." https://www.caida.org/data/monitors/passive-equinix-nyc.xml. Accessed: May 17th, 2021.

[12] L. Yuan, C.-N. Chuah, and P. Mohapatra, "ProgME: towards programmable network measurement," *IEEE/ACM Transactions on Networking*, vol. 19, no. 1, pp. 115–128, 2011.

[13] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "cSamp: a system for network-wide flow monitoring," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pp. 233–246, 2008.

[14] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proceedings of the 6th International Conference on emerging Networking EXperiments and Technologies*, pp. 1–12, 2010.

[15] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Sensing as a service model for smart cities supported by internet of things," *Transactions on emerging telecommunications technologies*, vol. 25, no. 1, pp. 81–93, 2014.

Chris Misa is a PhD student at the University of Oregon. His research interests include interfaces and systems for processing network traffic data streams. Chris's work has been recognized by several research fellowships including a Ripple graduate fellowship.

Ramakrishnan Durairajan is an Assistant Professor in the Department of Computer and Information Science at the University of Oregon. His research has been recognized with multiple NSF awards, Ripple faculty fellowship, UO faculty research award, several best paper awards, and has been covered in several fora.

Reza Rejaie is currently a Professor at the University of Oregon. He received a NSF CAREER Award for his work on Peer-to-Peer streaming in 2005 and a European Union Marie Curie Fellowship in 2009. Reza is a Fellow of IEEE (2017) and a Senior member of the ACM (2006).

Walter Willinger is Chief Scientist at NIKSUN, Inc. Before joining NIKSUN, he worked at AT&T Labs-Research and at Bellcore Applied Research. He is co-recipient of the 1995 W.R. Bennett Prize Paper Award, the 1996 W.R.G. Baker Prize Award, and of the 2005 and 2016 ACM/SIGCOMM Test-of-Time Paper Awards.