Special Issue on The ACM SIGMETRICS Workshop on Measurements for Self-Driving Networks

Arpit Gupta
University of California
arpitgupta@ucsb.edu

Ramakrishnan Durairajan
University of Oregon
ram@cs.uoregon.edu

Walter Willinger
NIKSUN, Inc.
wwillinger@niksun.com

The design and implementation of autonomous or "self-driving networks" represent some of today's most significant challenges in networking research. The vision for these networks is that they will be able to make management and control decisions in real time, typically without human intervention. Recent technological advancements, like SDN and 5G networks, along with scientific innovations such as XAI and transformers, have paved the way for this vision. Key innovations include: (1) fully programmable, protocol-independent data planes and the languages to program them; (2) scalable platforms capable of processing distributed streaming data, bolstered by the latest tools and software for data analysis and machine learning (ML).

A particularly promising development is the fusion of programmable control capabilities in the data plane with advanced ML-based inference techniques. This combination offers unprecedented opportunities for querying the network's state on a vast scale, providing the essential data for the many network management and control tasks that self-driving networks must autonomously perform.

However, the path toward realizing self-driving networks is strewn with obstacles. Practical, deployable system designs that are scalable and robust are scarce. Likewise, many ML-based inference tools available today are not production-ready; they typically lack generalizability, trustworthiness, or assurance of system safety. Realizing the vision of practical self-driving networks will require scalable system designs that employ closed-loop feedback at multiple levels to ensure their robustness with respect to the uncertainties of their environments. Moreover, a shift in perspective will be necessary for developing ML-based inference solutions. The success of the learning models that drive these solutions will have to be gauged by their explainability, trustworthiness, and safety rather than just traditional concerns like accuracy.

To assess the current level of interest and activity in this area, we organized the 1st Workshop on Measurements for Self-Driving Networks that took place in Orlando, Florida, USA, on June 19, 2023. The workshop was sponsored by NSF, organized by ACM SIGMETRICS, and held in conjunction with ACM SIGMETRICS 2023/FCRC 2023. This workshop served as a platform for researchers to present and discuss their latest research on technologies poised to make practical, deployable self-driving networks a reality. We sought contributions from experts in fields such as networking, applied as well as theoretical machine learning, network security, control theory, distributed systems, computer architecture, and data science, all united by their enthusiasm to realize the vision of self-driving networks.

The workshop featured presentations from invited speakers that represented 11 universities and included a diverse mix of senior and early-career researchers, as well as graduate students. All speakers were invited to submit a three-page paper on the topic of their presentation. The papers in this Special Issue are a testimony to the exciting ongoing developments in this area of research and address topics such as traffic monitoring, approximate querying, and decision making at data plane speeds; deployability and engineering challenges for self-driving networks; explainable network controllers; learning-assisted QoE enhancements; and the need for a paradigm shift in how ML-based solutions for networking problems in general and self-driving networks in particular ought to be developed and evaluated in the future so that they can be deployed and used in practice.

This collection of papers also highlights four fundamental limitations faced by researchers pursuing self-driving networks. These include: (1) a lack of capabilities to label network datasets at scale; (2) an urgent need for frameworks that facilitate privacy-preserving collaboration among researchers; (3) difficulties in developing provably generalizable ML artifacts; and (4) uncertainties about creating feasible pathways for safely road-testing ML models.

Technical discussions between speakers and participants during the workshop hinted at a potential path forward. In particular, one possible way to overcome these limitations involves building a community-wide infrastructure designed specifically to: (1) facilitate flexible, high-quality data generation and collection efforts that can be easily replicated across different networks; (2) offer an innovative framework for collaborative and privacy-preserving knowledge sharing, such as labeling functions, model specifications, and data features; (3) bolster a principled approach to developing generalizable learning models for networking problems; and (4) establish a strategy for deploying ML-based solutions in production networks.

We extend our gratitude to all the speakers, co-authors, and attendees who contributed with their presentations and actively participated in the workshop. We would also like to acknowledge the support of ACM SIGMETRICS and, in particular, the workshop co-chairs Leana Golubchik and Daniel Sadoc, and the PER editor Zhenhua Liu for their continuous guidance and assistance in producing this special issue of PER.

Workshop on Measurements for Self-Driving Networks Orlando, FL, USA

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

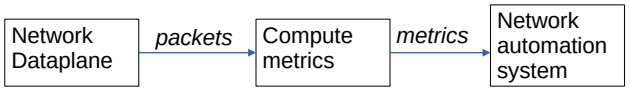


Figure 1: Traffic monitoring involves observing packets in the network and computing metrics for automation systems.

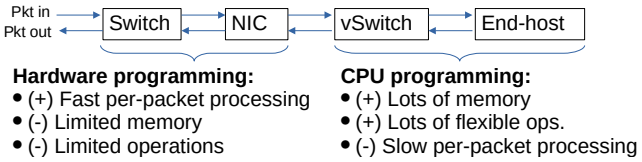


Figure 2: Processing platforms commonly considered in network traffic monitoring systems.

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(10MB) 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.

Making Decisions at Data Plane Speeds

Srinivas Narayana
Rutgers University, New Brunswick, NJ, USA

ABSTRACT

Feedback control loops to implement self-driving networks constitute *data collection* to sense the network, and *control algorithms* to make decisions driving the network. High-quality data is necessary for smart decisions. Yet, high-quality data is hard to obtain from the network data plane, due to insufficient visibility and large data volumes stemming from high packet rates. This paper distills principles to collect high-quality data arising from our own research experience: (i) filter and aggregate data as close to the source as possible; (ii) identify broad families of statistics that are measurable with bounded inaccuracy; (iii) don't assume low-level data plane software is easy to instrument, but instead (iv) apportion software flexibility by the time scales of the computation; and (v) prefer in-band approaches where possible for timely and efficient reactivity. We call the community to act upon these principles to leverage emerging opportunities using safely-extensible network stacks.

1. INTRODUCTION

Feedback control is an integral part of self-driving systems. Networks have conventionally incorporated feedback control at several layers of the stack to drive themselves. Classic examples include congestion control, medium access control, and IP traffic engineering. Feedback control includes two components: *data collection* to sense the network in the data plane, and *control algorithms* either in the data or the control plane, to drive the network based on the data that was collected.

Regardless of the smartness of decision-making algorithms, bad data can lead to poor decisions. Hence, it is paramount to have access to high-quality data from the data plane. However, there are several reasons why obtaining good data is challenging. To make our discussion concrete, we focus on scenarios in data center networks for the rest of this paper.

Why is it hard to collect high-quality data?

(1) *Insufficient visibility:* Designing feedback control to respond to anomalies in performance requires access to fine-grained, low-level network performance data directly measured at the bottlenecks. Examples include determining the queue lengths at routers and servers and the contributions of individual connections to those hotspots. However, such raw performance signals are often hard to measure in the data plane, because deployed hardware and software are simply incapable of the introspection required for such observations. The emergence of In-Band Network Telemetry on programmable dataplanes alleviates these problems to some extent, but it does not solve the visibility problem, especially for application-level metrics (§2.1).

(2) *Large data volumes.* When raw signals (e.g., queue sizes) are indeed available on a packet by packet basis, the speed at which such signals are generated poses a significant challenge. Naive attempts to collect such signals "out of band" using storage systems could double the number of packets processed by the network. Instead, either the per-packet signals must be sampled or aggregated to reduce the packet rate of outgoing signals, or bandwidth that could otherwise be used for actual network traffic must be repurposed to carry signals in-band, typically requiring server changes and new infrastructure. A related difficulty is the design of algorithms that can aggregate per-packet signals into useful "buckets" cutting across protocol layers, for example, organizing connections into a histogram of application-level response latencies. However, network data planes are traditionally only capable of simple stateful aggregations at a low protocol level.

Self-Driving Networks Workshop 2023 Orlando, Florida

2. LESSONS FROM THREE STORIES

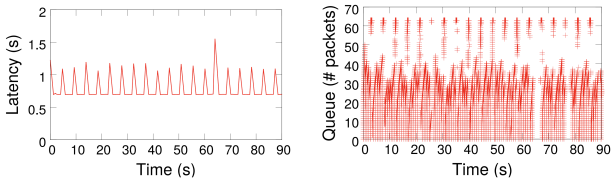
Given the difficulties of obtaining high-quality data, how should one go about designing algorithms to collect network data for self driving? In this section, we distill some principles from our own prior research efforts.

2.1 Network Performance Diagnosis

In the Marple system [3], our goal was to diagnose anomalies in network performance. An example of such an anomaly is *microbursts*: short-time-scale bursts of packets arising from traffic sources that display an ON/OFF transmission pattern, increasing the queueing delays transiently but recurrently for other latency-sensitive traffic sharing the network. Identifying the perpetrators of such microbursts is challenging, since they are not major contributors to traffic on the network and it is unclear which switch and queue in the network is the site of the microburst. At the time, the Tofino programmable switches had just introduced the capability to observe queue sizes as metadata on a per-packet basis on the switch pipeline. However, this raw data is arriving at the same high speed as the packets on the switch.

Principle 1 [Pushdown]. *Filter and aggregate data as close to the source of the data as possible.*

This principle is well known in database systems where the cost of moving data, say between machines or shuffling rows



(a) HTTP latency. (b) Queue sizes.

Figure 1: Microbursts: (a) A victim HTTP flow experiencing frequent spikes in response latency. (b) The time evolution of queueing delays experienced by packets traversing a queue with a microburst-perpetrating traffic source.

for database joins, is significant. Reordering database operations to eliminate irrelevant data earlier in the processing (e.g., matching on predicates or projecting specific columns) can significantly improve efficiency. Inspired by this principle, we designed primitives that operate directly in the switch data plane, at line rate, to implement filtering and aggregation through user-defined keys. For the microburst scenario, this enables the ability to (i) identify switch queues and packets which experience large queueing delays at those queues; and (ii) identify traffic sources (say, aggregated at the level of transport 5-tuples) that contribute ON/OFF traffic, by counting the number of bursts of packets separated in time by a user-defined threshold. This enables not only determining where the microburst-perpetrating sources are active, but also the sources themselves.

Principle 2 [Identify accurate families]. *Identify families of statistics measurable with bounded (or zero) inaccuracy, and design algorithms customized to those.*

The extensive literature on sketching algorithms adheres to this principle. However, it is much more generally applicable to data collection even for statistics not typically captured with sketches, for example, the number of packets considered out-of-order in a TCP connection. Concurrently with the Marple work, there existed hardware switching chips collecting aggregated network performance metrics, for example average packet latency per 5-tuple flow. However, when the switch experiences an uptick in the number of flows (e.g., under a flash crowd or a TCP SYN flood), memory size limitations would force the switch to evict existing flows from its memory. The policies used for eviction from the switch made it unclear how the data that is retained on the switch compares in accuracy to an ideal lossless measurement. However, the problem goes beyond the shortcomings of one platform: there was a fundamental lack of understanding of how a switch should collect measurements not easily summarized with limited memory.

In the Marple work, we identified a class of statistics for which it is possible to obtain accurate data despite the eviction of data from a switch under high memory pressure. The trick is that we use a slower, but larger and more persistent memory than a switch, to *merge* any partial measurements evicted from a switch with an authoritative measurement residing in the larger memory. A multi-tier memory architecture for measurement dovetails well with the existence of plentiful memory on servers outside of switches. We identified that statistics s whose per-packet update takes the functional form $s \triangleq \alpha(\vec{p}) \cdot s + \beta(\vec{p})$, where α and β can be any switch-implementable functions over a recent bounded history of packets $\vec{p}$, can be merged with 100% accurate results. This seemingly simple functional form captures diverse statistics, for example the number of out-of-order packets in each TCP connection.

2.2 Programming Congestion Control

Congestion control is a classic example of self-driving, with a rich research literature. In our work on the Congestion Control Plane (CCP [2]), we were inspired by the need to prototype and evaluate a complex congestion control protocol [1]—one that involves signal processing algorithms such as Discrete Fourier Transforms—in realistic settings.

Principle 3 [Low-level software changes slowly]. *Software is not arbitrarily fungible. In particular, data plane software is not easily changed, for reasons surrounding stability and performance.*

Traditionally, TCP congestion control is implemented using Linux kernel modules, which limit what developers are allowed to do. For example, floating point computations are challenging inside the kernel. Invoking some unsafe numerical operations could easily crash the kernel (e.g., division by zero). Further, the emergence of many kernel-bypass software platforms necessitated the implementation of the same protocol logic on diverse software platforms such as Intel's Data Plane Development Kit (DPDK) and Google's QUIC, each with their own relatively-static programming APIs.

In addition to asking if it is possible to ease development and experimentation for congestion control within Linux, we also wondered if it is possible to develop such logic once and have it run everywhere.

Principle 4 [Flexibility $\propto$ Available Compute Time]. *The flexibility accorded to a software layer should be proportional to the time available to compute at that layer.*

It was tempting to introduce a highly-flexible programming API to help develop complex functionality directly within the Linux kernel and emerging kernel-bypass frameworks. Specifically, the API could allow the maintenance of arbitrary state over which arbitrary computation could occur. However, high-speed packet processing is highly sensitive to the performance of the memory subsystem. For context, with 100Gbit Ethernet, it is necessary to admit a new minimum-sized Ethernet packet approximately every 6 nanoseconds to keep up with the packet arrival rate. In such contexts, the hit rates at the fastest cache layers are critical to performance—a single L2 cache miss could consume the entire time budget available to process a packet and slow the entire system down. Hence, the size of the memory maintained across packets must be limited, as should the compute over that memory. Not all complex functionality can or should go into the data plane.

In the CCP system, we observed that the nature of congestion control makes it neither necessary nor useful to implement complex congestion control computation for each packet in the data plane. Instead, the natural computational time scale for congestion control is the round-trip time (RTT) of the connection, which is much longer than the time to admit a new packet. Our design choice was to enable developers to write flexible yet simple *fold functions* in the data plane to maintain only the summaries of per-packet signals for each connection. These summaries would be relayed once every RTT to a much more flexible control plane component running in user space. The data plane

and the control plane components have asymmetric flexibility that is proportional to the natural time scales over which computations occur in those components.

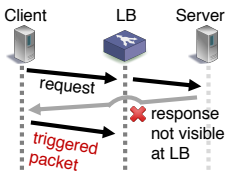
2.3 Server Load Balancing

In large-scale Internet services, it is standard practice to implement a layer of *server load balancing* to distribute the incoming workload of client requests across a pool of servers offering the service. The oft-stated goal of load balancing is to spread load, preventing hotspots or failures on any one server from impacting client requests. In our ongoing work on server load balancing [4], we ask whether load balancing could be used proactively to improve service performance, by redirecting more requests to the better-performing servers in the pool. Existing solutions that use server performance implement an *agent-based model,* where a software agent on the server (running either as a daemon or as a part of a library incorporated into the application) relays feedback on load and queue occupancies to the load balancer. Such explicit feedback is crucial since, in many deployments, the responses to clients from the servers skip the load balancer on the return path. This idea, known as *direct server return (DSR)*, significantly reduces the workload on the load balancer relative to processing both requests and responses.

With the advent of microservices, serverless, and nanoscale computing, there is now a move towards increasingly-finer granularity of computing per request. Shrinking compute times significantly hurt the usability of agent-based feedback. First, request processing becomes highly vulnerable to variability within the system, for example due to process scheduling. Second, server agents completely miss network delays. As the per-request compute time approaches the client connection's RTT, the network delay contributes half of the total client-visible response latency.

Principle 5 [In-Band Feedback Control]. *To design highly-reactive systems, avoid staleness and big data problems through in-band feedback control.*

Relevant data must be made available as quickly and as accurately as possible at the point where self-driving decisions are made. In performance-aware server load balancing, one relevant piece of data is an estimate of the up-to-date response latency offered by each server. Rather than siphoning data from server agents to load balancers through an out-of-band stream, or through a centralized data collection system, it is appealing if load balancers can measure the response latencies directly. However, the load balancer's visibility into client traffic is asymmetric: with DSR, the load balancer only sees the requests and not the responses, making it challenging to measure response latencies by correlating them with the requests.



Our key insight is that it is possible to substitute the measurement of the delay between request and response by the delay between the request and a packet that a client transmits due to the response—a packet we call a *causally-triggered transmission.* There are many examples of causally-triggered transmissions, most commonly TCP acknowledgments. We show that such transmissions can be detected while only observing requests but not responses [4].

3. A CALL TO ACTION

We believe there are significant opportunities ahead to design novel self-driving networked systems, by leveraging emerging *safely-extensible data plane* software in the network stack. Concretely:

1. *Verified kernel extensions* (eBPF) allow user-developed code to be attached with safety guarantees to specific "hooks" (function calls or execution sites) in the Linux kernel. Examples of hooks include the net device driver, packet scheduler, congestion control, and system calls. Safety in the eBPF context means that programs have a bounded running time, contain only instructions that do not crash (e.g., no division by zero), and all memory accesses are within safe bounds permitted by the kernel.
2. *Service proxies* (e.g., Envoy, Linkerd) are a new software layer in the container networking stack, refactoring common communication-related capabilities needed in containerized applications into a reusable component. For example, service proxies implement load balancing policy and failure detection and recovery logic common to multiple applications. Some service proxies such as Envoy are safely extensible at run time, including WebAssembly (WASM) sandboxes.

These extensible software layers enable the design of novel algorithms for data collection and feedback control operating directly in the packet-processing software path, with well-designed channels to communicate out-of-band with a flexible control plane. For example, one could incorporate application-specific customizations for congestion control, packet scheduling, or high-speed packet forwarding. Extensible data plane software is naturally amenable to applying principled data collection and feedback control techniques (§2) that overcome the fundamental challenges of data collection (§1). We believe that the prospects of designing self-driving networks have never before been as bright.

However, to make those prospects viable, the community must address wide-ranging challenges to enable the effective use of these emerging extensible network layers.

(1) Designing algorithms under safety constraints: Any program run within an extensible network layer must be 'safe'—a term whose definition depends on the context (e.g., which kernel version and which hook are we extending?), and is evolving. This brings up questions like: What is the scope of algorithms that can be implemented safely within extensible network layers? What programming abstractions could make it easy to design such algorithms?

(2) Performance: Achieving high performance within extensible software layers is crucial since these are on the critical path of packet processing. How should the performance of an algorithm be optimized while retaining its safety guarantees? How should we design optimizing compilers? Is there scope for workload-driven optimizations?

We call upon the community to act on these challenges to help realize novel self-driving networked systems.

4. REFERENCES

[1] Prateesh Goyal et al. Elasticity detection: A building block for internet congestion control. In *SIGCOMM*, 2022.

[2] Akshay Narayan et al. Restructuring endpoint congestion control. In *SIGCOMM*, 2018.

[3] Srinivas Narayana et al. Language-directed hardware design for network performance monitoring. In *SIGCOMM*, 2017.

[4] Bhavana Vannarth Shobhana et al. Load balancers need in-band feedback control. In *ACM HotNets*, 2022.

Toward Fast Query Serving in Key-Value Store Migration with Approximate Telemetry

Alexander Braverman
Seven Lakes High School
Katy, Texas

Zaoxing Liu
University of Maryland
College Park, Maryland

ABSTRACT

Distributed key-value stores scale data analytical processing by spreading data across nodes. Frequent migration of key-value shards between online nodes is a key technique to react to dynamic workload changes for load balancing and service elasticity. During migration, the data is split between a source and a destination, making it difficult to query the exact location. Existing solutions aiming to provide real-time read and write query capabilities during migration may require querying both source and destination servers, doubling the compute/network resources. In this paper, we explore a simple yet effective measurement approach to track the key-value migration status, in order to improve the query-serving performance under migration. In our preliminary prototype, we use a Bloom filter on the destination server to keep track of individual key-value pairs that have been successfully migrated. For key-value pairs that have yet migrated, the information stored in the Bloom filter enables fast forwarding to the source server without the need to check the database. We prototype this design on a local cluster with Redis deployments. Our preliminary results show that this approximate measurement-based design minimizes query losses during migration.

1 Introduction

Modern cloud services (e.g., e-commerce, mobile gaming, and social networking) depend on large-scale key-value stores as the backend to perform various kinds of jobs (e.g., content caching, real time analytics and machine learning) [16, 3]. These services often require backend databases to process requests over ever-growing data volumes and dynamic workload distributions. However, static sharding limits the ability of such systems to adapt to rapidly changing workloads. This can result in degraded performance and Service Level Agreement (SLA) violations due to load imbalance and insufficient provisioning of cloud resources [8, 9].

To tackle the problem of imbalance of load and resources, a variety of key-value migration techniques are adopted [8, 9, 12] to efficiently migrate data between nodes (i.e., the source and destination servers). However, the migration process itself is often time-consuming, and the actual query serving performance varies depending on the workload distribution and the migration progress. During migration, the client is unknown about which keys have reached their destination (migrated) at any given time without actually accessing them on the relevant databases. Therefore, it is difficult for the client to always query certain keys at the *right* location because the client is unsure of the current location of the queried key. To ensure query serviceability, a straightforward solution is to query both the source and destination servers, incurring doubled overheads for the client. Alternatively, the client needs to access a database that records the migrated keys, but such maintaining such a database is resource-heavy (e.g., network, compute, and storage), since the number of keys can be prohibitively large. Ideally, the client should know where to access the queried key in the right location without expensive bookkeeping and without actually reaching the key-value store hosted on the servers during the migration process.

To this end, we explore and leverage approximate telemetry approaches to track the status (e.g., "not started", "in transmission", or "completed") of the key-value pairs during migration, allowing client requests to be served at the right location as soon as possible. Approximate measurement design brings a major benefit to serving client queries in key-value store migration: the underlying data structures used in the design are often probabilistic and require only sublinear resources, such as sketches [7, 14] and Bloom filters. Such resource savings enables wider adoption of approximate telemetry on resource-constrained devices. Using these devices to track migration status, including SmartNICs [1] and programmable switches [5], can direct client requests to the appropriate location as early as possible. Thus, in this preliminary work, we deploy a Bloom filter on the Redis server to track migration progress and evaluate potential query-serving performance improvements.

We implement a key-value migration protocol with a Bloom filter deployed on the destination server. We deploy a large Redis key-value store [2] (100GB) to migrate from one commodity server to another. We evaluate several experimental scenarios with client workloads following Zipf distributions [13] with varied write ratios in client requests. Our preliminary results demonstrate that using the Bloom filter to track key-value migration status can significantly reduce query mishits (defined as query loss rate).

2 Preliminary Design and Prototype

Our workflow to perform key-value migration consists of four main components: (1) the migration process of Redis key/value pairs from a source server to a destination server, (2) a Bloom filter that identifies whether a key-value pair has migrated successfully and if a query is missed in the local database, the filter tells the client where to for-

ward the missed requests, (3) a client that sends read and write requests to the destination server, and (4) a forwarding mechanism which allows requests to be served before the respective key reaches the destination server. Below, we provide the details of each component using Redis [2] as an example.

- **Migration**: The migration process [9] consists of extracting all the key/value pairs from a Redis instance of the source server and sending them to the destination server and storing them in its Redis store. To allow integration of Bloom filters, we reimplement a UDP-based migration protocol to extract Redis key value pairs from source to destination by extending the implementation of DistCache [13].

- **Bloom filter**: During the migration process, when an individual key-value pair reaches the destination server and is updated in the Redis store, this key will be added to a Bloom filter [4] on the destination side to keep track of which key-value pairs have already successfully migrated. Employing a simple Bloom filter at the destination has two benefits: (1) It does not have false positives and can be adjusted to achieve relatively low false negative rates as shown in Figure 1. No false positives ensure that queries to the migrated keys will never go back to the source server. (2) The memory efficiency of the Bloom filter makes it possible to serve as a "cache". This additional saving of space comes from the fact that we only store keys and not values in the bloom filter.

- **Serving client requests**: When the client queries certain key/value pairs currently in the destination, the destination server will first check whether this key has reached the server by probing the Bloom filter. The advantage of using a probabilistic filter over directly accessing Redis to see if a key-value pair has already migrated is that the Bloom filter is small in space and allows for faster (parallel) memory accesses. If the key has migrated, we know for sure that it is in the filter, and we can access the Redis store locally and directly respond to the client.

- **Request forwarding**: If a key is queried by the client, which is not already in the Bloom filter, we know it still has not reached the destination server. Therefore, the destination server can simply forward [6] the client request to the original sending server, who will be able to serve the request on its own local Redis instance and respond directly to the client.

3 Experiments

In our evaluation, every experiment conducted consisted of using 50 million Redis key-value pairs for migration, which held 100 GB of data. Our experiments run on a testbed consisting of 3 servers, each of which has two Intel Xeon Gold 5317s, 512GB DRAM, and a 10G NIC. Specifically, each key-value pair held 2000 bytes, the keys ranging from $1 - 50,000,000$, and the values being a randomly generated string with 1992 bytes to 1999 bytes of data depending on the length of its respective key. Every experiment we run generates 750,000 queries from the client. Furthermore, we use different Zipfian distributions for our experiments, such as .9, .95, and .99. Second, we also use different write ratios, such as .01 to .05. We evaluate the loss rates when the queries go to the wrong locations without being forwarded to the right places for Bloom filter-based vs. no filter-based.

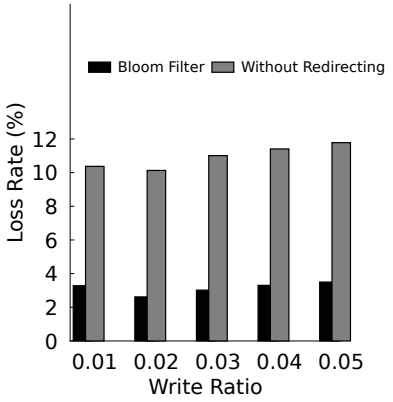


Figure 1: Query loss rates with varying write ratios.

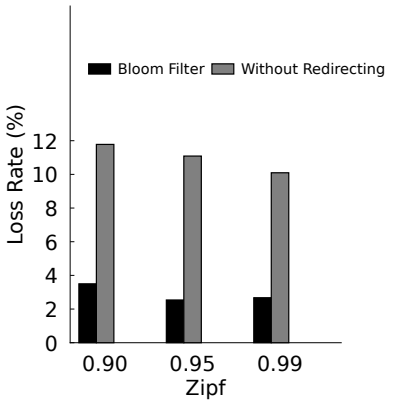


Figure 2: Query loss rates with varying Zipf distributions.

To evenly split the migration data and serve client requests in parallel, we used 10 threads for both the source and destination servers. Moreover, the data on the destination server is randomly split amongst 5 Redis servers, to mitigate the loss due to the large volume of migrated data. On the destination server, the Bloom filter implementation [17] we used has 125 MB of data stored overall in the data structure.

3.1 Effect of Query Distribution

Our first experiment revolves around the effect of the write ratio of queries to the loss rate. This experiment on average takes 14 minutes and 39 seconds across all write ratios. The client issues key read/write queries following a Zipfian distribution of 0.9 skewness with write ratios of 0.01 to 0.05 as shown in Figure 1. The inclusion of Bloom filter and request forwarding demonstrates a 3.5 × improvement in mitigating query loss on average (from 3.162 to 3.870).

3.2 Effect of Write Ratio

Our second experiment measures the effect of how the skewness in Zipf distribution affects the loss rate. This experiment on average takes 14 minutes and 38 seconds across

all Zipf distributions tested (0.9, 0.95, 0.99) and we use the write ratio of 0.05. As shown in Figure 2, the inclusion of the forwarding of requests results in a 3.837 × improvement in mitigating query loss on average (from 3.367 to 4.367).

3.3 False Negatives in Bloom Filter

One of the main concerns of using bloom filters is the false negatives. In our key-value migration case, false negatives occur when the key is shown to be in the Bloom filter, but in reality it has yet been migrated to the destination. In each experiment, we calculate the number of false negatives by checking when the Redis server returned *null*. In our experiments, the number of such false negatives is always less than 0.043 percent of all queries. Thus we can conclude that with a relatively large filter (e.g., 150MB), false negatives have negligible impacts on the average query-serving performance.

3.4 Evaluation Summary

Overall, these experimental results demonstrate the need for designing a proper measurement approach to using forwarding, as without the bloom filter, there is a notable drop in performance that could prove costly in a real setting. Furthermore, as the results demonstrated the positive effect of forwarding, using higher power devices such as SmartNICs [1, 15] could prove to be even more beneficial [10] in serving client queries during migration.

4 Discussion

We conclude by highlighting a subset of new opportunities for further research in this space.

Approximate telemetry for key-value migration on near-user programmable devices. Our prototype demonstrates the benefits of tracking the detailed migration status with a Bloom filter-based design. However, for simplicity, the filter for tracking the migration progress is deployed on the server side. Every client request needs to pay the cost of reaching the server first before it can be directed to the right location. We posit that, with emerging flexibility and programmability in the network devices (e.g., SmartNICs and programmable switches), we can find a vantage point in the network to easily measure how key-value pairs are being migrated while not being far from the client.

Adaptive filters for improving the false negative rates. The current prototype is limited to the standard Bloom filter with fixed false negatives. While our experiments show small performance degradation on average, the false positives can incur performance problems at the tail (e.g., tail query latency for some keys). Recent advances in adaptive filters [11] have shown some practical designs to dynamically tune the filters to reduce and control the error rates if we have seen such false negatives so far.

5 References

[1] Bluefield data processing units. `https://www.nvidia.com/en-us/networking/products/data-processing-unit/`.

[2] Redis. `https://redis.io/`.

[3] Redis use cases. `https://redis.com/blog/5-industry-use-cases-for-redis-developers/`.

[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, jul 1970.

[5] 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.

[6] J. Chen, P. Druschel, and D. Subramanian. An efficient multipath forwarding method. In *Proceedings. IEEE INFOCOM '98*, pages 1418–1425, 1998.

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

[8] J. Kang, L. Cai, F. Li, X. Zhou, W. Cao, S. Cai, and D. Shao. Remus: Efficient live migration for distributed databases with snapshot isolation. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2232–2245, 2022.

[9] C. Kulkarni, A. Kesavan, T. Zhang, R. Ricci, and R. Stutsman. Rocksteady: Fast migration for low-latency in-memory storage. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 390–405, 2017.

[10] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. V. Lakshman. Uno: Uniflying host and smart nic offload for flexible packet processing. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, page 506–519, New York, NY, USA, 2017. Association for Computing Machinery.

[11] D. J. Lee, S. McCauley, S. Singh, and M. Stein. Telescoping filter: A practical adaptive filter. *arXiv preprint arXiv:2107.02866*, 2021.

[12] Y.-S. Lin, S.-K. Pi, M.-K. Liao, C. Tsai, A. Elmore, and S.-H. Wu. Mgcrab: transaction crabbing for live migration in deterministic database systems. *Proceedings of the VLDB Endowment*, 12(5):597–610, 2019.

[13] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica. Distcache: Provable load balancing for large-scale storage systems with distributed caching. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 143–157, 2019.

[14] 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, 2016.

[15] H. Seyedroudbari, S. Vanavasam, and A. Daglis. Turbo: Smartnic-enabled dynamic load balancing of µs-scale rpcs. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1045–1058, 2023.

[16] R. K. Singh and H. K. Verma. Redis-based messaging queue and cache-enabled parallel processing social media analytics framework. *The Computer Journal*, 65(4):843–857, 2022.

[17] T. Wang. Integer hash function. `https://gist.github.com/badboy/6267743`, 2007.

Tackling Deployability Challenges in ML-Powered Networks

Noga H. Rotman
The Hebrew University of Jerusalem

1 Introduction

Following the success of Machine Learning (ML) in various fields such as natural language processing, computer vision and computational biology, there has been a growing interest in incorporating ML into the networking domain [5, 6, 14, 4, 9]. Today, ML-based algorithms for prominent networking problems such as congestion control, resource management and routing, perform very well when their training environment is faithful to the operational environment, achieving state-of-the-art results when compared to traditional algorithms. However, the adaptation of these algorithms to function in production environments has not been straightforward, as real-world networks may differ greatly from the data used for training, leading to a drop in performance when unleashed into the wild.

This paper provides an overview of the problems impeding the successful deployment of ML-powered networks. It categorizes proposed solutions into three types, based on the main concern they address. Notably, each category takes place at different stage of the lifecycle of an ML-powered network: in-training, pre-deployment, and online, allowing to employ all three in tandem. We propose a holistic approach to tackling the challenges facing a successful deployment, by intervening at *all* three stages, and integrating the observations obtained at each stage to improve the others.

1.1 Example: ML-based algorithm in the wild

ML-based networking protocols have been very successful when their training environment and test environment match. What happens when this is not the case?

To demonstrate the impact to performance that may occur when the training environment of an ML-powered algorithm differs from the deployed environment, we consider a deep learning solution to the timely problem of adaptive video streaming (ABR) in HTTP-based video streaming.

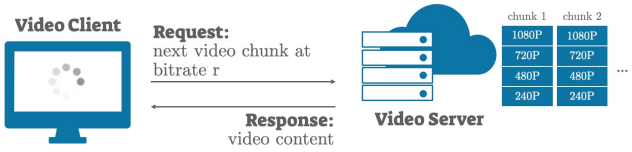


Figure 1: ABR problem overview

In ABR (see Figure 1), a client interacts with a video server. Videos are stored on the server as chunks of (roughly) the same length, each chunk available in different bitrates, corresponding to its quality. When the client is watching a video, they need to request the next chunk at a specific bitrate r. The ABR algorithm, running at the client, is responsible for choosing which bitrate r should be requested from the server. The algorithm must walk a thin line between selecting a lower resolution that may not be satisfactory to the client, and choosing a higher one that may force the client to wait for content while streaming (an event also known as rebuffering), as both effects are known to play an instrumental role in user engagement [1].

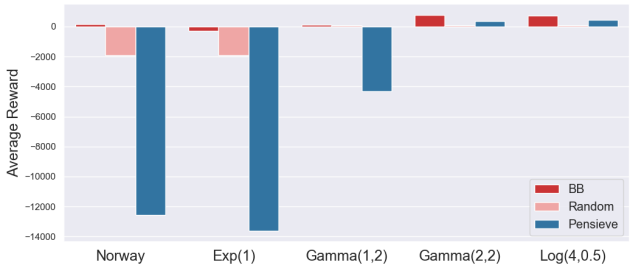


Figure 2: Average reward of an ML-based algorithm, Pensieve, trained on a dataset collected in Belgium, compared to that of BB, a non-learning algorithm, and a random algorithm. When tested on a dataset collected in Norway and two synthetic datasets, the Pensieve agent performs worse than the random algorithm. Figure 2a from [10].

Pensieve [6] applies Reinforcement Learning (RL) [11] to ABR. In [10], we trained Pensieve agents on different datasets, both real-world and synthetic. These agents performed well when the test and training environments were drawn from the same distribution. We then tested their performance when the training and test environments differs. Figure 2 contrasts the performance of a Pensieve agent trained on a dataset collected in Belgium, against two algorithms: Buffer-Based (BB) [3], a manually-crafted and widely deployed protocol, and an algorithm which chooses the next bitrate randomly. In all cases, the performance of BB is better than the one of exhibited by the Pensieve agent. Furthermore, in some cases, such as a test set collected in Norway, the Pensieve agent performed *worse than the random algorithm*. These results raise concerns when considering deploying ML-powered networks.

1.2 The causes for the impact to performance in the real world

There are multiple factors contributing to the drop in performance of an ML-based algorithm in a networking production environment.

First, bad generalization is a known trait of several ML methodologies prevalent in networking, such as RL.

Second, one must consider the usage and evaluation of these algorithms. Typically, when a non-learning algorithm fails, the cause is traceable, as the algorithm's logic is clear. It is then possible to adjust it to better handle the networking conditions causing the failure. On the other hand, when an ML-based algorithm breaks down, the reason is obscured, as neural networks are painfully difficult for humans to understand. Because of this, not only are we unable to effectively investigate the cause of failures, but we are also unable to modify the algorithm to address them.

Third, modern communication systems are architecturally complex and extremely dynamic. Encompassing all possible scenarios in a training set is simply not possible, as deployment environments are too diverse.

2 Tackling deployability challenges

In this section we classify proposed solutions for mitigating the performance degradation caused by bad generalization into three categories, based on the main concern they target (see Section 1.2), and provides a short overview of each. Further discussion of these categories can be found in Section 3.

In-training enhancements target the components responsible for the initial creation of an ML-based algorithm: the data used for training, the actual training process, or both. Successfully adapting ML methodologies to the networking domain is an arduous task, as the problems these techniques were originally created for and tested on differ greatly from networking problems.

Pre-deployment analysis aims to eliminate potential problems prior to deployment by providing insights as to the actions taken by a trained ML-based networking protocol, and the reasons leading to them.

Online assurances attempts to intervene during deployment, in order to avoid a sudden drop in performance when network conditions change.

2.1 In-training enhancements

Puffer [13] is an online service streaming live U.S. TV stations. Born as a research project, it serves as a playground for evaluating and comparing various ABR protocols. The authors proposed a new ML-based algorithm to the ABR problem, where the learning agent is trained daily *in-situ*, using data obtained from its deployment environment during the last fourteen days. Puffer won the NSDI'20 community award, as the data collected is made available online.

Another notable example is Genet [12], which targets the generalization problem in *RL-based* networking protocols by focusing the training procedure on the most challenging environments, instead of choosing them uniformly at random. To do so, the authors use Curriculum Learning [8]. While this methodology has proved useful in other domains, applying it in a networking context is nontrivial, as it is unclear how to measure the "difficulty" of a network environment.

2.2 Pre-deployment analysis

In [7], the authors introduce two categories of ML-powered networking systems: *local* and *global*. Their framework, Metis, translates the trained neural network employed by a local or global system into either a decision tree or hypergraphs. Both of these representations are much easier for humans to understand and evaluate. Interestingly, this approach advocates for the deployment of the *representation* in place of the original ML-based algorithm, thus allowing to modify the algorithm running in production directly. It is worth noting that there are ML-powered networking systems that cannot be addressed using this formalism.

Formal Verification is a mathematical approach for reasoning about a neural network's behavior. It provides provable guarantees of specified requirements; for example, for ABR, one can ascertain that when the conditions of the network do not allow for high average quality, the algorithm opts for a lower resolution over constantly rebuffering. This approach is used in [2] to evaluate three proposed ML-based networking protocols. A known disadvantage of this approach is that it is hard to scale to larger neural networks. In networking, however, most ML-based algorithms involve relatively small neural networks, making this approach feasible for various ML-based networking algorithms.

2.3 Online assurances

The last technique aims to rein in the possible costs of bad generalization in a production environment by replacing the ML-based algorithm with a "safer" option, when the decisions of the former are incoherent/uncertain. The motivation for this methodology is simple: in communication networks, there are many hand-crafted protocols that have been deployed in various networks for years, some for decades. While these algorithms may not enjoy the high performance achieved by ML-based algorithms, they were designed to withstand disastrous circumstances, and are thoroughly tested "in battle". Therefore, if we were able to successfully identify *online* when an ML-based algorithm is making incoherent decisions, we would be able to provide a kind of a "safety net", by enabling to switch to a "safer" option once such a need arises. Further incentive can be found in Figure 2, demonstrating that on test sets in which the Pensieve agent fails to generalize, BB could potentially be used to improve the overall performance.

Realizing this technique requires addressing several major challenges. First, although the detection of uncertainty in the behavior of an ML-based algorithm has been explored in different contexts, no standard method has yet to emerge. Second, translating an uncertainty signal into a measurable amount that can be calculated *online*. Third, forming heuristics by which to set thresholds on the calculated value, in order to determine whether the system should switch to a "safe" algorithm.

We presented this concept in [10]. We investigated three possible signals: uncertainty in the algorithm's input, uncertainty in the actions selected, and uncertainty in the algorithm's evaluation of its future benefit from the chosen actions. We tested these signals and compared their impact on Pensieve agents. We have found that two of the tested signals show promise.

3 Discussion and conclusions

Despite encouraging advances, realizing the promise of ML-powered networks is still elusive. This paper discussed the challenges encountered during deployment of these systems, summarized current techniques, and offered a fresh perspective of these practices.

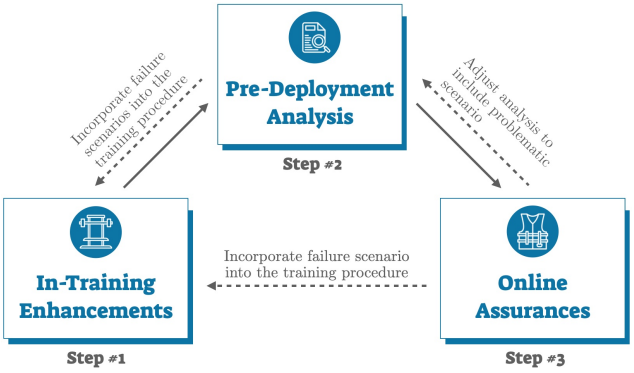


Figure 3: Each category takes place at a different phase, allowing not only to utlize all three, but continuously enhancing the algorithm based on its observable behavior at each step.

A key insight of this paper is that the categories presented in Section 2 are *complimentary*, as each occurs at a different stage of the ML-powered network pipeline: during the initial training of the algorithm, pre-deployment, and online. We claim that in order to enable the successful deployment of ML-powered networks, we must address the problems at *all* three stages (see Figure 3). First, in-training enhancements should be applied, creating a more resilient algorithm. Then, pre-deployment analysis of the resulting algorithm should be performed, which may lead to further enhancements of the training procedure and data. Finally, the system should be adjusted to provide online assurances, so that when the algorithm eventually fails in production, the reasons can be investigated and resolved while avoiding a critical hit to performance.

This approach essentially calls for a paradigm shift when considering these algorithms, from a "one shot" scenario to an iterative process, in which the algorithm is re-examined, re-adjusted and re-tested prior to and during deployment.

4 Acknowledgments

Many thanks to the organizers of the ACM SIGMETRICS '23 Workshop on Measurements for Self-Driving Networks at Orlando, Florida, on June of 2023, for their invitation, and to the participants of the workshop, for the engaging discussion. Special thanks to Dr. Yotam Feldman.

5 References

[1] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. *ACM SIGCOMM computer communication review*, 41(4):362–373, 2011.

[2] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 305–318, 2021.

[3] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[4] Nathan Jay, Noga H. Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019.

[5] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56, 2016.

[6] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210, 2017.

[7] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 154–171, 2020.

[8] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020.

[9] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. *DOTE*: Rethinking (predictive) *WAN* traffic engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1557–1581, 2023.

[10] Noga H Rotman, Michael Schapira, and Aviv Tamar. Online safety assurance for learning-augmented systems. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 88–95, 2020.

[11] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[12] Zhengxu Xia, Yajie Zhou, Francis Y Yan, and Junchen Jiang. Genet: automatic curriculum generation for learning adaptation in networking. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 397–413, 2022.

[13] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, 2020.

[14] Francis Y Yan, Jestin Ma, Greg Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. Pantheon: the training ground for internet congestion-control research. *Measurement at http://pantheon. stanford. edu/result/1622*, 2018.

Engineering Autonomous Self-Driving Networks

Mariam Kiran
Oak Ridge National Laboratory
1 Bethel Valley Rd
Oak Ridge, TN, USA
kiranm@ornl.gov

ABSTRACT

Networking infrastructure, e.g. wide area networks (WAN) connecting data centers worldwide, or wireless 5G and beyond, are all witnessing unprecedented traffic demand, due to massive data-explosion in software applications involving text, image and video transfers and demand for high quality connectivity. The infrastructure itself is often limited by budget; and providing optimum performance with limited resources such as high bandwidth or low latency for user quality experience, is a challenge. Continually upgrading to expensive high-end switches and optical fibers is not a long-term feasible solution.

Artificial intelligence (AI) approaches are quickly gaining popularity as a means to build self-driving networks. Incorporating AI into the network middleware can help manage itself, making intelligent decisions based on current demands, resource availability and handle multiple distributed network devices efficiently. In early 2000s, IBM introduced the autonomic design with self-x properties for any system to become 'smart'. In this article, we discuss AI being leveraged to develop the four self-x properties for networks.

Keywords

Autonomous, self-driving network, machine/deep learning

1. INTRODUCTION

Science innovation and discovery is using high performance computing, heterogeneous hardware, high-speed and low latency connections like 5G and the advent of quantum computing are massively impacting the experiment and their data production rates. Particularly for distributed experiments, data movement requires high-speed and optimum network connections that can present experiments with their data needs - large or small transfers, high-speed connections and minimum loss, in data intensive experiments from high energy physics, climate sciences, biomedical research, the Large Hadron Collider (LHC) [2], to name a few. This presents unprecedented challenges, like global data movement, optimizing compute and storage resources and much needed network innovation. As a means, machine learning have proven successful in networks such as network traffic prediction [8], optimizing routing calculations [6] and finding security anomalies in flow traffic. Similarly, software defined networking (SDN) and virtualization (NFV) have rapidly evolved network research towards programmatic control and automated configuration concepts and technologies. This is a large deviation from previously labor-intensive network deployments.

Acting as essential middleware to advanced computing facilities, networks need to cater to software application complexities and sometimes move petabytes of data at high-speeds for fast processing and storage systems. With increasing infrastructure complexity, real-time processing demand and fast bulk data transfers, current networking hardware is finding it hard to cope with the need to satisfy users and industry alike. One way of upgrading the communicating links and middleware involves adding advanced switches or routers that perform fast processing and trace packets as flows move across the network processing massive amounts of data [1]. However, this comes with drawbacks, for example, SDN has demonstrated traffic optimization over physical links based on demand, but is slow with processing overhead and too much data being analyzed into one central decision-making module [4]. Additionally, most devices are not SDN-enabled and can only configure specific devices [9].

Upgrading to high-end switches and redesigning network ecosystems, decoupling network control from devices, can be expensive and introduces new problems of vendor lock-in, high maintenance and management problems across globally distributed sites.

Machine learning algorithms can be used to predict network behavior such as 'which path selection, capacity or QoS change will cause what result or event X with what probability P'. Detecting anomalies will cut down costs and time spent finding impaired segments or misbehaving devices in network infrastructures. Actively forecasting traffic demands can allow engineers to anticipate congestion, and proactively perform better capacity planning for continued reliable connectivity for critical experiments. Making these decisions in near real-time requires sufficient data processing power to rapidly digest relevant traffic datasets as time series, factors that affect the application performance and build algorithms that can improve and optimize network behavior. The ML/AI methods that can be used in network research include supervised and unsupervised classification, regression and reinforcement learning. However, in networking, it is often difficult to find labeled data sets, as performance logs are rarely labeled except in major event scenarios. As an example, unsupervised classification can recognize good flow performance or security anomalies [3]. Reinforcement Learning (RL) is a type of machine learning that allows an agent to interact in an environment by trial

and error, and use feedback on its actions and experiences to learn optimal behavior. Allowing agents to learn under real conditions in a network is tricky, and the lack of demo platforms has limited progress in this area.

With AI advances, there are innovative ideas to combine network logs, SDN and current network configuration, to make networks smart and self-managed ecosystems. This AI phenomenon can vastly improve and solve traffic optimization problems by predicting anomalies in real-time, forecast utilization and eventually repair itself, when faults such as when hosts fail or packet loss occurs. However, this idea of *smart network* is not new. In early 2000s, IBM outlined the autonomic computing middleware [5] concepts, which allowed computing middleware components such as virtual machine manager, job scheduler and clusters to become autonomic or have the *self-x* properties.

2. AUTONOMIC ARCHITECTURE

Autonomic describes self-management. This is not *autonomous* or *automatic*, where devices behave on their own or follow a predefined script. Autonomic devices follow human-directed goals, but interpret them locally depending on their own capability and environment. Human administrators have little direct influence and the devices can self-regulate themselves using high-level policies. IBM introduced the autonomic computing initiative [5] which argued to use autonomy principles focused on developing concepts that allow systems to self-correct and operate independently. These systems are built to follow adaptive behavior, perceive changes from the environment, reason and correct themselves automatically.

Autonomic networking follows the same concepts of autonomic computing, that can handle increasing complexity by self-regulating its components using high-level policies. Also called the *self-x* properties, these are (1) Self-configuration: Automatic configuration of components. (2) Self-healing: Automatic discovery of anomalies and fault correction. (3) Self-optimization: Automatic monitoring and resource optimization to ensure optimal functions of resources, given high-level requirements are met. And (4) Self-protection: Proactive identification and protection from attacks. Technically, self-x behavior can be achieved by any device by monitoring (observe its own state and behavior), reasoning (analyze and decide changes) and controlling (perform changes) itself. IBM's MAPE architecture (monitor, analyze, plan and execute) as a knowledge engine, which the device uses to make intelligent decisions).

Table 1 shows how the critical components of self-x behaviors. Using multiple AI techniques for individual network elements provide a novel approach to manage a distributed and large system, which is different from traditional bespoke vendor design specifications. Figure 1 shows a network element, such as router learning from the network environment and using a reward function to optimize its behavior.

3. AI IN NETWORK RESEARCH

Typically, operators monitor link traffic quantitatively, measuring data moving across the router interfaces and using their *experience* to estimate traffic volumes and surges. If congestion is anticipated, operators reroute traffic through alternate network links by reprogramming routers (or traffic reengineering) which usually takes between 8-24 hours to program, compile, and deploy. However, accurately predicting traffic congestion time is a complicated task for two main reasons: (1) Network traffic displays very random behavior and does not show any seasonal patterns (2) Monitoring all links in large WAN architecture (typically greater than 100) is not a feasible solution. Statistical prediction methods such as ARIMA and Holt-Winters have been successful in network traffic prediction. Additionally, deep learning solutions, like LSTM-based neural network architectures, can learn features during the training phase.

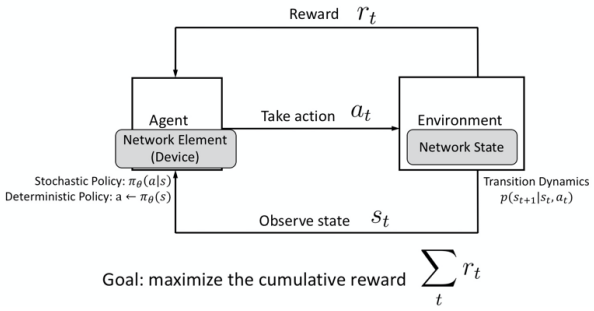


Figure 1: A network agent using reinforcement learning to observe, plan and execute actions.

Novel techniques from deep learning [8] have shown to use graph neural networks, transformers and advanced ML methods to provide a active prediction system, that predict network congestion with recent data, providing high accuracies to make real-time decisions. However, changing routing based on congestion prediction and how this impacts networks is yet to be seen in practice.

Similarly, self-learning controllers have used deep reinforcement learning to learn and optimize network routes [6]. These are innovative new techniques using real-time streaming monitoring data to make just-in-time decisions on how to move large or small flows on the network. However, we find that even though AI provides many innovation, there is a lack of hardware (e.g. controllers) that can input these AI decisions to realize the change on the network. These warrant new areas of research to develop self-driving networks.

4. FURTHER WORK

AI can optimize the network at various levels, however, there are some general research areas still open.

Scattered Network Logs: Network engineering collects multiple various data sets scattered with multiple network statistics. Merging these to build knowledge graphs to understand general application-network relationships is a challenging task. Recent work [10] uses multiple network monitoring tools and big data models to find relationships between multiple diverse variable sets, and merge them into one dashboard. Such systems could be key to generate training data sets, useful for feeding the learning model with subtasks. Adding techniques such as clustering, unlabeled pattern recognition and dimension reduction will be integrated to reduce the problem domain of Big Data processing computationally, and not compromising on the deduced training sets created. This includes pioneering and sustaining architectures to optimize infrastructure for end-to-end data movements, network providers deal with multiple users (onsite users, remote users), collects monitoring data on utilization, energy consumption, bulk data transfer, routing,

Self-X Capability	Current networking capabilities	Autonomic networking	AI-enabled opportunities
Self-configuration	Networks are labor intensive and statically configured. It is impossible to respond to dynamic changes in traffic and network changes.	Using high-level policies or intent, components can be configured to achieve the system goal and dynamically adapt to changes.	Understanding high-level intent via knowledge representation and natural language processing can help elements understand the goals of the autonomic element.
Self-optimization	Current usage of resources is statically set with tuning parameters. For example usage of links is decided by the OSPF protocols.	Continuously monitoring and adjusting resource to improve performance and QoS can help optimize how resources are being utilized. Such as underutilized links.	Using unsupervised / supervised machine learning methods to extract feature and behavior patterns can help autonomic elements understand performance.
Self-healing	Diagnosing faults and failing links can take hours and involves network operators pouring over several systems logs to find the problem.	Networks can monitor systems logs and identify abnormal behavior and initiate fault management and repairing strategies for countering failing nodes.	Time-series analysis.
Self-protection	Detecting and recovering from potential attacks can take hours or weeks. Certain alarm systems can be set to alert engineers about potential problems on the network currently happening.	Networks can detect malicious attacks by analyzing time-series data sets, to find anomalous behaviors and automatically initiate repair and fault management systems.	Time series analysis

Table 1: Comparison of autonomic and AI-enabled capabilities

flow dynamics and performs monitoring. For example traffic Data with SNMP counters based on tagged interfaces, monthly ingress traffic per router, flow data such as NetFlow for IPv4 or IPv6, topology data for link metrics, security logs or even perfSONAR for network diagnostics, packet loss, host-level and service level checks, I/O speed can also be used to make decisions.

Computational Processing of Big Data: Studying the right features and grouping can help determine parameters that best support network traffic patterns and infer behavior. Prior work [7] minimized computational cost by porting SNMP data processing onto cloud clusters using Hadoop and real-time processing. With each feature, measurable objectives can help quantify them (e.g. uptime expressed as percentage, or requests per application, system throughput and operational cost). This also helps identify groups to summarize features for application needs to create application profiles and QoS demands.

Building the Right Objective Functions: There are two range of objective functions - Short-term and long-term analysis objective functions. For example, optimizing current traffic patterns based on different times of the day, or optimizing loads, loss and time schedules are short-term goals. But where regression techniques can learn traffic demands over larger time periods to build a network model, isolate repeated failing paths with performance declines are good for long-term network analysis. In order to optimize both, focusing on optimizing SDN with path-based reasoning along with improving QoS and application intents can be key.

We argue that autonomic and self-driving are interchangeable terms when building a autonomic network. This term is also not the same as autonomous systems which are completely independent systems. This achievement has a potential to enable a new research domain and a collection of various ranges of autonomic networks that can work together with other networks for user and application objectives.

5. REFERENCES

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[2] L. R. Evans. *The Large Hadron Collider: a marvel of technology. EPFL* Press: Fundamental Sciences, 2009.

[3] S. Hao, A. Kantchelian, B. Miller, V. Paxson, and N. Feamster. Predator: Proactive recognition and elimination of domain abuse at time-of-registration. In *ACM SIGSAC*, CCS '16, page 1568–1579, 2016.

[4] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford. Optimizing bulk transfers with software-defined optical wan. SIGCOMM '16, 2016.

[5] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1), Jan 2003.

[6] M. Kiran, S. Campbell, and N. Buraglio. Hecate: Ai-driven wan traffic engineering for science. In *SC INDIS Workshop*, pages 41–49, 2022.

[7] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja. Lambda architecture for cost-effective batch and speed big data processing. pages 2785–2792, 2015.

[8] T. Mallick, M. Kiran, B. Mohammed, and P. Balaprakash. Dynamic graph neural network for traffic forecasting in wide area networks. In *IEEE Big Data*, pages 1–10, 2020.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[10] B. Mohammed, M. Kiran, and B. Enders. Netgraf: An end-to-end learning network monitoring service. In *SC INDIS Workshop*, pages 12–22, 2021.

Towards Future-Based Explanations for Deep RL Network Controllers

Sagar Patel[1], Sangeetha Abdu Jyothi[1,2], and Nina Narodytska[2]

[1]University of Califoria, Irvine [2]VMware Research

ABSTRACT

Lack of explainability is hindering the practical adoption of high-performance Deep Reinforcement Learning (DRL) controllers. Prior work focused on explaining the controller by identifying salient features of the controller's input. However, these feature-based methods focus solely on inputs and do not fully explain the controller's policy. In this paper, we put forward *future-based explainers* as an essential tool for providing insights into the controller's decision-making process and, thereby, facilitating the practical deployment of DRL controllers. We highlight two applications of future-based explainers in the networking domain: online safety assurance and guided controller design. Finally, we provide a roadmap for the practical development and deployment of future-based explainers for DRL network controllers.

1. INTRODUCTION

Deep Reinforcement Learning (DRL), in lab settings, offers state-of-the-art performance in increasingly more problems in the networking domain, such as load balancing, network traffic engineering, congestion control, and adaptive bitrate streaming. However, DRL controllers lack real-world deployment because operators cannot interpret, debug, or trust them [5].

The domain of eXplainable Reinforcement Learning (XRL) has emerged to address this lack of trust. At its core, XRL aims to explain the decision-making process of a learned controller to humans [1]. Prior work has interpreted the controller's actions by highlighting the important features given to the controller. Metis [5] applies the concepts of decision tree distillation and critical path identification to generate interpretations. Trustee [3] further builds on the process of distillation by introducing ways to improve fidelity and generating an associated trust report. We broadly categorize these works as *feature-based*.

Feature-based explainers have proven their effectiveness in a number of applications. They can identify issues with the feature set [3], dataset [3], and model architecture [5]. However, they do not capture the forward-looking objective of the controller's decision-making process and thus cannot provide a comprehensive understanding of the controller.

In this work, we present a new perspective on explainability that we define as *future-based*. This approach focuses on presenting a future-oriented perspective of the controller by capturing goals or rewards. In networking applications, future rewards, in particular, are human-designed and represent the key performance metrics of the network application. In this case, future-based explanations based on reward components can provide meaningful insights into future performance metrics, which are meaningful to network operators. For instance, by analyzing future rewards in congestion control, we can obtain insights into the upcoming performance of the controller in terms of throughput, latency, and loss.

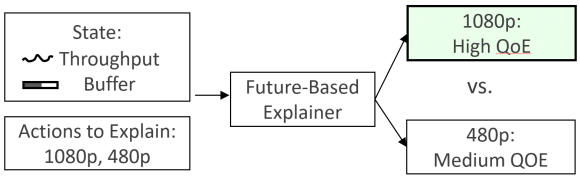
We highlight two key benefits of gaining a future-based understanding of DRL network controller behavior. First, it can provide insights for fine-tuning the algorithm parameters and the reward function during DRL controller design, which is a tedious and resource-inefficient process. Second, during the practical deployment of DRL controllers, future-based explainers can enable online safety assurance [6] by supporting network observability and preemptively triggering alerts for upcoming performance declines.

Recent work has introduced future-based explainers for gaming and robotic environments [4, 2]. However, these solutions are not adopted in practice since they either require accurately modeling the environment or require significant changes to the controller, which is often not feasible or detrimental to controller's performance. We outline the key research challenges towards developing practical future-based explainability frameworks in the networking domain. First, the forward-looking view of the controller contains a vast amount of information; capturing it succinctly and precisely is crucial for practical adoption. Second, future-based explainers must have low-latency explanations to spot safety violations before they happen, which is critical for tasks like online safety assurance. Third, they must function separately from the controller. This ensures they can be broadly applied without making extensive changes to the controller that could negatively affect the performance. Fourth, the explainers should be robust to malicious attacks, noise, and distribution shifts, thereby avoiding a false sense of security.

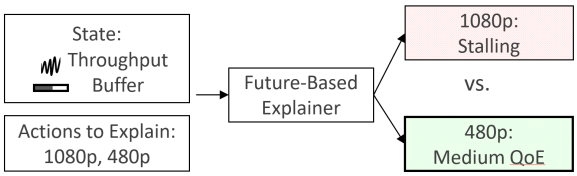
2. BACKGROUND

In this section, we provide a background of Reinforcement Learning and Adaptive Bitrate Streaming.

Reinforcement Learning. In Reinforcement Learning, an agent interacts with an environment. It is given a state s_t, and takes an action a_t according to its policy $\pi(A|s_t)$. The environment reacts to the agent's action and gives back to it the reward r_t, along with the next state s_{t+1}. The goal of the agent is to change its policy such as to maximize the reward over time, defined as the return $G = \sum_{t=0}^{\infty} \gamma^t r_t$.



(a) A contrastive future-based explanation for actions within state S_1



(b) The contrastive explanation for actions within state S_2

Figure 1: We illustrate how future-based explainers can provide insights across states and actions. We consider two seemingly similar states, S_1 and S_2, and seek to understand why the controller prefers different actions in them. We query the explainers with two actions: 1080p and 480p under both state S_1 (a) and S_2 (b). We can then peek into the future impact of these actions under both states and understand that 1080p is preferred in S_1 because it leads to high QoE, but it is not preferred in S_2 because it causes stalling.

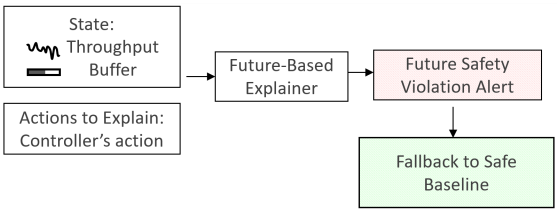


Figure 2: Online Safety Assurance: With the ability to capture the future performance of the controller, future-based explainers can be used to raise alerts about safety violations before they occur, falling back to a safe baseline and guaranteeing tail-ended performance.

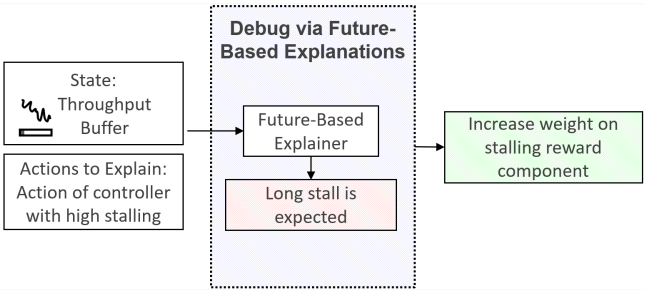


Figure 3: Guided Controller Design: Future-based explanations can help tune controller design. For a controller acting aggressively under poor network conditions, these explanations reveal stalling as an expected outcome. This insight helps the operator see that the reward function needs greater penalties for stalling.

Adaptive Bitrate Streaming. Adaptive Bitrate Streaming (ABR) works by dividing the video into chunks and encoding them at various discrete bit rates. During streaming, the most suitable bit rate for each chunk is chosen based on network conditions. The client also has a short buffer that can hold chunks yet to be seen. The ABR controller sequentially selects the bitrate to maximize the client's Quality of Experience (QoE), a numerical measure that awards high quality, and penalizes both changes in quality and stalling.

3. FUTURE-BASED EXPLAINERS

In this section, we first provide an overview of future-based explainers and how they can concretely enhance explainability. Next, we highlight two key applications of future-based explainers in the networking domain towards facilitating practical deployment of DRL controllers.

3.1 Overview

Future-based explainers shed light on the future goals or performance of the DRL controller. To train a future-based explainer, we take three inputs: the DRL controller, the simulation environment, and the training traces. We then roll out the controller, collecting the states, actions, and rewards the controller gets while interacting with the simulation environment. Finally, we use this interaction data to train the future-based explainer.

During inference, we can query the future-based explainer with a state and action to obtain a view into the impact of that action—getting explanations built around future states or rewards.

As a concrete example, let us consider a future-based explainer of an ABR controller that captures future performance through rewards and a scenario where the operator is looking to understand why the controller chooses different actions under seemingly similar network conditions. The operator selects the states representing this scenario and queries the explainer for the different actions it takes. In Figure 1, we visualize this scenario. We want to understand why the controller prefers to send a 1080p video chunk in S_1 while a 480p chunk in S_2, despite both of them having unstable throughput and similar buffer occupancy. We query a future-based explainer with both actions in both states. We can see that in S_2, the 1080p action is likely to lead to stalling and is thus avoided. Meanwhile, because the same is not expected in S_1, the 1080p action provides a higher quality of experience.

Thus, future-based explanations offer a medium to compare the impact of different actions from within and across multiple states.

3.2 Applications

Next, we discuss how insights offered by future-based explanations can be leveraged in the design and deployment of DRL controllers. We highlight two key applications: guided controller design and online safety assurance.

Guided Controller Design. Implementing practical DRL solutions demands various design choices. These range from selecting the feature set, picking the DRL algorithm and its hyperparameters to designing the reward function and learning parameters. Tuning of these design parameters is

a tedious and resource-inefficient process in practice, even for DRL experts. Typically, these parameters are tuned through a trial-and-error process.

Future-based explainers can aid in this design process. They offer insight into the exact factor the DRL algorithm optimizes: future performance. To demonstrate their utility in identifying DRL algorithm issues, we examine an example. In Figure 3, we debug an aggressive controller under poor network conditions. Using a future-based explainer, we find that the controller, despite anticipating long stalls due to its actions, still opts for them. This discovery suggests to the operator the need to increase the penalty for the stalling reward component.

Online Safety Assurance. Online Safety Assurance poses the challenge of detecting when the learning-based policy is likely to reach an unsafe state and avoiding it by falling back to a reliable and extensively tested baseline policy. This fallback mechanism acts as a "safety net" for learning-based systems, designed to facilitate high-performance outcomes under ideal circumstances while also offering minimum performance guarantees under less than perfect conditions [6]. Existing research has suggested that the problem can be addressed by quantifying uncertainty within the learning policy [6], where uncertainty serves as an indirect measure of potential unsafe states. This has been accomplished either through an ensemble method or novelty detection.

In this context, future-based explainers can be applied directly to foresee and alert for possible unsafe behavior without the need for a proxy. Figure 2 illustrates how such a system would work. The future-based explainer would receive the current state as input, predict the controller's future behavior, and issue warnings for potential safety violations. In response to these warnings, a fallback to a safe baseline can be triggered. This ensures the overall system maintains compliance with safety requirements and performance commitments.

4. KEY CHALLENGES

In this section, we describe the main research challenges in developing practical future-based explainers.

Concise Explanations. Future-based explainers must create their explanations by considering the future: a series of states, actions, and rewards. The challenge lies in converting this complex information into a format that humans can easily understand.

Low-Latency Inference. To support real-time applications such as online safety assurance, future-based explainers must provide explanations promptly. Moreover, this process should not disrupt the primary operations of the controller. In short, generating explanations must add minimal cost to the controller's decision latency. Fortunately, future-based explainers can offer insights beyond a single step in the future. Thus, they can function in parallel without being on the critical path of the controller.

Separation from the Controller. To ensure broad applicability, future-based explainers should not require significant modifications to the controller. Such changes can harm the controller's performance, introducing a performance and explainability trade-off. Instead, explainers should leverage the controller's inner workings, such as its learned features, without altering them.

Robustness of Explanations. The ability to create explanations that are robust to malicious attacks, noise, and shifts in distribution is a significant, unresolved challenge. Notably, even several widely-used feature-based explainers have proven susceptible to these threats [8]. However, building on early intuitions [7] and addressing this issue is critical to support trust-sensitive applications such as online safety assurance.

5. CONCLUSION

In this paper, we present an initial perspective on a new angle of explainability with future-based or forward-looking explainers. We highlight their ability to power guided controller design and enable online safety assurance. We then provided a road map for practically implementing future-based explainers by detailing key open research challenges. We envision this work to lay the foundation for a broad application of future-based explainers.

6. REFERENCES

[1] Nadia Burkart and Marco F Huber. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70:245–317, 2021.

[2] Francisco Cruz, Richard Dazeley, Peter Vamplew, and Ithan Moreira. Explainable robotic systems: Understanding goal-driven actions in a reinforcement learning scenario. *Neural Computing and Applications*, pages 1–18, 2021.

[3] Arthur S Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A Ferreira, Arpit Gupta, and Lisandro Z Granville. Ai/ml for network security: The emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1537–1551, 2022.

[4] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on explainable artificial intelligence*, 2019.

[5] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 154–171, 2020.

[6] Noga H Rotman, Michael Schapira, and Aviv Tamar. Online safety assurance for learning-augmented systems. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, pages 88–95, 2020.

[7] Dylan Slack, Anna Hilgard, Sameer Singh, and Himabindu Lakkaraju. Reliable post hoc explanations: Modeling uncertainty in explainability. *Advances in neural information processing systems*, 34:9391–9404, 2021.

[8] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 180–186, 2020.

Enabling Perception-Driven Optimization in Networking

Yihua Cheng, Xu Zhang, Junchen Jiang (University of Chicago)
{yihua98,zhangxu,junchenj}@uchicago.edu

ABSTRACT

Service providers struggle to catch up with the rapid growth in bandwidth and latency demand of Internet videos and other applications. An essential contributor to this resource contention is the assumption that users are equally sensitive to service quality everywhere, so any low-quality incidents must be avoided. However, this assumption is not true. For example, our work and other parallel efforts have shown that more video users can be served with better quality of experience (QoE) if we embrace the fact that the QoE's sensitivity to video quality varies greatly with the video content. To unleash such benefits, the application systems must be driven by not only system measurement data but also user feedback data that capture users' perceptions of service quality. In this short paper, I will highlight some of our recent efforts toward the efficient collection of user feedback and enabling perception-driven optimization for Internet applications.

1. INTRODUCTION

The landscape of online applications has seen a sea change over the past few years, with multiple trends driving up the bandwidth demands for online videos. The rise of ultra high-definition (4K/8K/VR) videos dramatically increases the per-video bandwidth demand and is projected to be 22% of global video traffic in 2022 from 3% in 2017. Video traffic to mobile devices has also more than tripled in the last 3 years. The rising bandwidth demands widen the gap between user expectation and user-perceived *quality of experience* (*QoE*) measured in mean opinion score or user engagement.

Achieving better bandwidth-QoE tradeoffs relies on accurate QoE models. Widely used in modern video delivery systems, a QoE model takes a streamed video (such as buffering stalls and visual quality index, etc) as input and returns a predicted QoE as output. Most adaptive-bitrate (ABR) and CDN/ISP resource allocation algorithms (*e.g.,* [9, 6]) use QoE models to predict when increasing video quality has more QoE improvement. Thus, any errors of a QoE model can mislead these optimization techniques to pick suboptimal decisions and miss opportunities to improve QoE or save bandwidth.

Indeed, recent efforts have shown that the sensitivity of QoE to these optimizations differs significantly across videos, web pages, and even across different segments of the same video (*e.g.,* [11, 2]). Therefore, having more accurate QoE measurements allows content providers to strategically allocate more compute/bandwidth resources or enhance quality at points of higher QoE sensitivity (see §2). With these trends, QoE measurements are increasingly needed.

While there have been many efforts to make QoE measurements faithfully reflect true user experience, relatively less attention has been given to *building a system that obtains QoE measurements fast.* Two relevant efforts exist—one automates QoE measurements by using crowdsourcing and the other uses collected QoE measurements to dynamically prune videos that no longer need QoE ratings. Unfortunately, it is challenging to combine the two ideas, because with the existing crowdsourcing interface, one must specify which videos to be rated by how many users *before* each crowdsource task begins, making it hard to dynamically prune redundant videos without launching multiple crowdsourcing campaigns. In short, prior work suffers from two limitations: *(i)* The speed to obtain QoE measurements is still quite slow due to the traditional crowdsourcing interface; and *(ii)* QoE measurements can be obtained for only on-demand content, not live content.

This short paper introduces two projects[1] that aim at addressing these limitations. First, to speed up QoE crowdsourcing, we have *developed and open-sourced*[2] *VidPlat, the first re-usable tool for fast and automated QoE measurements.* VidPlat allows dynamic pruning of QoE video samples in one single crowdsource task. To realize it, VidPlat creates a new shim layer between the researchers and the crowdsourcing platform, allowing researchers to define a logic that iteratively creates new videos that need more ratings based on the latest QoE measurements. Compared to existing QoE measurement methods, VidPlat (1) keeps all QoE measurements in one crowdsourcing task, thus minimizing the overhead to initialize tasks and re-calibrate/train raters, (2) dynamically decides when enough ratings are gathered for each video, thus reducing the total number of QoE ratings, and (3) is an open-source platform that future researchers can re-use and customize.

Second, to enable the QoE measurements on live videos, we present SensitiFlow, an alternative architecture that *online* profiles and adapts to quality sensitivity by continuously gathering and analyzing QoE-related feedback from real video sessions watching the same video. Concretely, SensitiFlow orchestrates the adaptive-bitrate (ABR) logic of video sessions. SensitiFlow runs an online *feedback loop* with two components. At a high level, SensitiFlow maintains the quality-sensitivity profile of each video segment by continuously collecting QoE-related feedback from ongoing video sessions, which can be average user ratings or whether a higher percentage of users quit/skip watching a video segment under low quality than under high quality. Based on the up-to-date quality-sensitivity profiles, SensitiFlow makes ABR decisions to improve QoE for concurrent and future video sessions watching the same video.

2. WHY IMPROVING EFFICIENCY OF QOE MEASUREMENTS

Since it is hard to directly ask users to rate their subjective

[1] `https://people.cs.uchicago.edu/~junchenj/perception_driven_optimization`
[2] `https://github.com/orgs/QoEStudies/repositories`

experience in real-time, researchers and content providers run offline user studies to assess QoE under various objective quality metrics. Participants are asked to watch an application *demo*. In video QoE, a demo can be a video streamed with a one-second buffering stall deliberately added at a certain point. In web QoE, a demo can be a web page loaded with a certain page load time (*e.g.,* a certain above-the-fold time). Then the participants rate the subjective QoE score in the range from 1 to 5. Finally, we can calculate the mean QoE scores of each demo video and model the relationship between QoE and quality metrics.

Potential of QoE-driven optimization: Traditionally, QoE models are expected to capture the *general* relationship between QoE and a few quality metrics. As a result, once enough QoE measurements are collected to model QoE on several representative videos or web pages, the QoE models will be re-used on other videos or pages. However, many recent efforts have shown that more granular, *context-specific* QoE models, which quantify the QoE-quality relationship of *individual* video (or even video segments) [11]. The shift from one-size-fits-all QoE models to context-specific QoE models quickly increases the frequency and amount of QoE measurements. For instance, Netflix produces on average more than 580 minutes worth of new video content every day. If it builds a separate QoE model for each minute of video, it will ask 10 raters to watch and rate 100 hours of videos every day. Similarly, web QoE research also shows a similar increase in the demand for QoE measurements [2, 3]. These context-specific QoE models can substantially improve QoE without using more bandwidth or compute resources. For example, in video streaming, applying per-video QoE models to adaptive bitrate (ABR) algorithms in video players can improve 15.4% QoE without using more network bandwidth [11]; in web services, we can have 40% QoE improvement by allocating computing resources across different web requests by their QoE models [3, 12]. More QoE measurements are also needed when a new optimization (*e.g.,* a new video bitrate ladder or chunk segmentation [7]) is proposed whose impact on QoE may not be captured by existing QoE models.

Related work and limitations: As the need for QoE measurements rises, so does the need to reduce the *latency* (to recruit workers and collect QoE ratings) and the *cost* of QoE measurements (total compensation given to the workers who provide the QoE ratings).

Two efforts exist to reduce the delay and cost of QoE measurements. First, several efforts (*e.g.,* [5, 10]) have shown the potential of automating QoE measurements using **crowdsourcing platforms** like Amazon Mechanical Turk (MTurk) and Prolific. These works have been focused on retaining reliable crowd workers, calibrating QoE ratings, mitigating hidden confounders (*e.g.,* order of assignment completion or different user devices), and reducing cost via dynamic pricing. Second, depending on the QoE ratings already collected, many demos would be **redundant** and can be **pruned** to let participants rate fewer demos [8]. For instance, to investigate how video bitrate affects the QoE of a particular video, if human raters are unable to perceive the QoE difference between bitrates of 1 Mbps and 10 Mbps, then no ratings will be needed for the bitrates between 1 Mbps and 10 Mbps on this video.

A natural question then is *will QoE measurement be automated and made much faster as promised by these approaches?* Unfortunately, the answer is no, because to dynamically prune demos, researchers have to *sequentially launch a series of crowdsourcing tasks* and use the QoE measurements from one task to decide which demo can be pruned in the next task, causing significant delays.

3. VidPlat: A PLATFORM FOR FASTER QOE CROWDSOURCING

To fully realize the speed benefit of crowdsourced QoE measurements, we have developed *VidPlat, the first re-usable open-source tool that enables dynamic demo pruning to speed up crowdsourced QoE measurements.* VidPlat serves as a shim layer between the researchers and the crowdsourcing platforms. VidPlat launches one task, but unlike the traditional crowdsourcing interface that requires researchers to pre-determine the demos and the number of QoE ratings per demo upfront, VidPlat offers a more flexible interface to researchers. Researchers are allowed to define a logic and a few initial demos, and upon receiving a QoE rating, VidPlat invokes this logic to determine the subsequent demos based on the logic's output. Then instead of immediately showing the next demo to raters, the next demo will be first put in a queue, and VidPlat will decide which demo in the queue should be given to the next rater. Using this "indirection" between which demos need more QoE ratings and which demo to be rated next by a worker, VidPlat retains the flexibility to randomize the order demos seen by a rater and avoid asking a worker to rate too many (similar) demos.

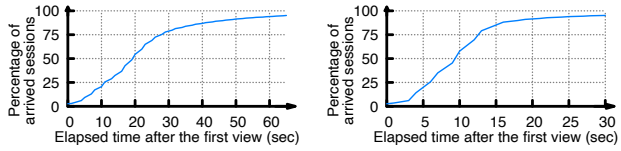
In short, with VidPlat, researchers do not need to determine all the demos or the required number of QoE ratings before the user study task begins; instead, VidPlat lowers the development burden while still collecting crowdsourced QoE measurements with minimum redundancy. As a result, it greatly reduces the number of demos and QoE ratings collected, thereby saving both time and cost.

Use cases: VidPlat has already been used in three IRB-approved QoE-related projects: *(i)* investigating the relationship between webpage load time and QoE [12]; *(ii)* exploring the correlation between video quality and QoE in on-demand video streaming [11]; and *(iii)* comparing the QoE impact of video bitrate and motion-to-photon (MTP) latency in online video gaming [4]. VidPlat's dynamic assignment determination significantly improved the efficiency of our user studies. For instance, compared to Sensei [11], a prior tool employing a traditional interface, VidPlat reduced both costs and latency by more than 50% in these use cases, while obtaining QoE models that realize the same QoE improvement as Sensei. These empirical results demonstrate the tangible benefits of our novel approach.

4. SensitiFlow: ENABLING QOE-DRIVEN OPTIMIZATION IN LIVE VIDEOS

Though VidPlat speeds up QoE measurements, the content of the video must be known *beforehand*. How to estimate the variability of quality sensitivity in *live* videos? One may use heuristics, such as VMAF or the popularity of a video segment to infer quality sensitivity, but how sensitive users are to quality under different content is often more complex than what can be captured by these heuristics [11].

So can we use real users to profile quality sensitivity, especially in live videos? Fortunately, this is feasible since most views of a live video segment occur dur-



(a) A (live-linear) TV show (b) A live-event sports video

Figure 1: *Example arrival patterns of views of the same live video segment: 20% of sessions watch the same content at least 3 seconds earlier than 60% of sessions.*

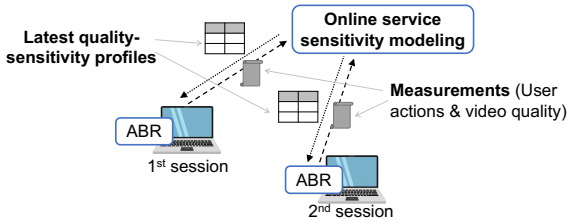


Figure 2: *Each session in SensitiFlow uses the latest quality-sensitivity profile to make ABR decisions and updates the global coordinator with the latest user actions.*

ing a non-trivial time span of 30-40 seconds after it first being viewed. Figure 1 shows the relative wall-clock time of sessions watching a chunk of a live-linear video (24/7 live programs [1] such as talk shows, TV plays) and a live-event video (*e.g.,* live sports broadcasting). Our conversation with domain experts has confirmed that such time discrepancies among live video viewers are commonly accepted in the industry. Over the last decades, the modest time difference (10-30 seconds) among viewers has become an accepted feature (rather than a bug) of live internet videos and efforts to realize full synchronicity in large-scale live events have been lukewarm due to the implementation complexity.

Inspired by this observation, SensitiFlow's *global coordinator* (depicted in Figure 2) constantly collects online measurements of per-segment quality and QoE-related feedback (*e.g.,* exit or skip) from video sessions to maintain an up-to-date view of the *quality-sensitivity profile* of each video. A quality-sensitivity profile maps each segment and quality level to the estimation and variance of quality sensitivity, in engagement drops and retention drops. It can answer the "what-if" question: *what would the expected drop in engagement/retention for a given quality at each segment?* When a session's ABR logic decides the bitrate of the next video segment, it will query the quality-sensitivity profiles and make ABR decisions using logic such as the one described in [11].

To test the gains of SensitiFlow on user engagement (QoE), we consider a simple logic that works in two phases. In the *profiling phase*, the first N sessions use a *default ABR logic*, and their per-segment user engagement and quality metrics are collected and used to estimate the quality sensitivity of the segment. After N sessions, it enters the *optimization phase*, in which each session runs a variant of the quality-sensitivity-aware ABR algorithm proposed in [11]. The algorithm takes as input the player's current state (history throughput, buffer length, etc) and the quality sensitivity of the next three segments and returns as output ABR decision for the next chunk. We evaluated QoE in user engagement (view time) using real traces of 7.6 million video sessions from a content provider. Our preliminary results show that SensitiFlow can realize up-to 80% of the improvement obtained by a hypothetical "oracle" system that knows quality sensitivity in advance.

5. VISION: USER-CENTRIC NETWORKING

SensitiFlow and VidPlat show the early promise of a more *user-centric* approach, where *measurements on user experience and actions* are first-class citizens of system monitoring and optimization. Just like systems metrics indicate current system states, user actions and engagement reveal individual user's experiences, as they watch a video, browse a web page, or use a mobile app. While we study only video systems in this paper, we think that the general approach may be applicable to other network applications such as gaming and mobile web. For instance, users' tolerance to web page loading time is better modeled by directly observing users' natural actions (*e.g.,* [12, 10]). This user-centric approach calls for novel system designs to realize the *tight* control loop between (near-)real-time user experience measurement and system adaptation. SensitiFlow and VidPlat take a step in this direction, and working toward such a perception-driven system is an active direction of future research.

6. REFERENCES

[1] Linear Live Streaming 101. `https://antmedia.io/linear-live-streaming-101/`.

[2] M. Butkiewicz, D. Wang, Z. Wu, H. V. Madhyastha, and V. Sekar. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *NSDI*, 2015.

[3] P. Casas, S. Wassermann, N. Wehner, M. Seufert, and T. Hossfeld. Not all web pages are born the same content tailored learning for web qoe inference. In *2022 IEEE International Symposium on Measurements & Networking (M&N)*, pages 1–6. IEEE, 2022.

[4] Y. Cheng, A. Arapin, Z. Zhang, Q. Zhang, H. Li, N. Feamster, and J. Jiang. Grace++: Loss-resilient real-time video communication under high network latency. *arXiv preprint arXiv:2305.12333*, 2023.

[5] T. Hoßfeld, M. Hirth, J. Redi, F. Mazza, P. Korshunov, B. Naderi, M. Seufert, B. Gardlo, S. Egger, and C. Keimel. Best practices and recommendations for crowdsourced qoe-lessons learned from the qualinet task force" crowdsourcing". 2014.

[6] J. Jiang, S. Sun, V. Sekar, and H. Zhang. Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation. In *USENIX NSDI*, 2017.

[7] M. Licciardello, L. Humbel, F. Rohr, M. Grüner, and A. Singla. [solution] prepare your video for streaming with segue. *Journal of Systems Research*, 2(1), 2022.

[8] X. Liu, W. Song, Q. He, M. Di Mauro, and A. Liotta. Speeding up subjective video quality assessment via hybrid active learning. *IEEE Transactions on Broadcasting*, 69(1):165–178, 2022.

[9] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with pensieve. In *ACM SIGCOMM*, 2017.

[10] M. Varvello, J. Blackburn, D. Naylor, and K. Papagiannaki. Eyeorg: A platform for crowdsourcing web quality of experience measurements. In *CoNEXT*, 2016.

[11] X. Zhang, Y. Ou, S. Sen, and J. Jiang. Sensei: Aligning video streaming quality with dynamic user sensitivity. In *USENIX NSDI*, 2021.

[12] X. Zhang, S. Sen, D. Kurniawan, H. Gunawi, and J. Jiang. E2e: embracing user heterogeneity to improve quality of experience on the web. In *ACM SIGCOMM*. 2019.

A NetAI Manifesto (Part I): Less Explorimentation, More Science

Walter Willinger
NIKSUN, Inc.

Arpit Gupta
UCSB

Arthur S. Jacobs
UFRGS

Roman Beltiukov
UCSB

Ronaldo A. Ferreira
UFMS

Lisandro Granville
UFRGS

ABSTRACT

The application of the latest techniques from artificial intelligence (AI) and machine learning (ML) to improve and automate the decision-making required for solving real-world network security and performance problems (NetAI, for short) has generated great excitement among networking researchers. However, network operators have remained very reluctant when it comes to deploying NetAI-based solutions in their production networks, mainly because the black-box nature of the underlying learning models forces operators to blindly trust these models without having any understanding of how they work, why they work, or when they don't work (and why not). Paraphrasing [1], we argue that to overcome this roadblock and ensure its future success in practice, NetAI *"has to get past its current stage of explorimentation, or the practice of poking around to see what happens, and has to start employing tools of the scientific method."*

1. INTRODUCTION

Most deployed networking solutions, be they ubiquitous protocols such as TCP or special-purpose systems such as load balancers, make decisions based on domain-specific heuristics that rely on partial network state information extracted from active or passive network measurements. For more than two decades, networking researchers have been exploring how to improve and automate these heuristics-based decision-making processes with the help of NetAI. In the process, they have enthusiastically embraced the development of new learning models by applying a workflow paradigm commonly referred to as the "standard ML pipeline." Comprised of *(i)* a learning task that is characterized by a model specification, *(ii)* a training dataset, and *(iii)* an independent and identically distributed (iid) evaluation procedure, this paradigm provides a blueprint for producing trained models that "work." Here, the statement "the model works!" is short for "according to the evaluation procedure used, the model has excellent expected predictive performance (*e.g.,* F1-score close to 1) when used for the originally posed learning task."

Like in many other application domains of ML (*e.g.,* computer vision, self-driving vehicles), leveraging this workflow paradigm in the networking domain has also enabled transformational progress, with ML-based solutions frequently and easily outperforming domain-specific state-of-the-art heuristics. However, despite this progress and ensuing promises, NetAI in its current form has largely failed to gain traction among network operators.

In this paper, we criticize the use of the standard ML pipeline that is popular with NetAI researchers. In particular, we show that relying on this widely-adopted ML workflow is fraught with problems that question the scientific foundations of the artifacts it produces and argue for abandoning it altogether in favor of a new generation of ML pipelines. In the process, we elaborate on the urgent need to be able to develop ML models that are either inherently explainable or can be explained post-hoc by applying available global explainability tools. We describe an initial attempt at designing and implementing such a new ML pipeline that is capable of accomplishing this feat, comment on its ability to aid the development of a new generation of learning models that focus on the generalizability and safety of ML models, and discuss some exciting new opportunities that arise as a result of this proposed paradigm shift in ML model development and evaluation.

2. THE "DUMBING DOWN" OF NETWORKING RESEARCH

The main reason why ML-based solutions have not been widely adopted in networking is that the models that the standard ML pipeline outputs are in general black boxes. In effect, such an output forces network operators to blindly trust the resulting learning model, providing them with little to no understanding of how the model works, why it works, or when it doesn't work (and why not).

This dissatisfaction has been compounded by an increasing awareness among researchers that the standard ML pipeline defines indeed a low bar for claiming that the trained models it produces as output "work." In particular, by relying on an evaluation procedure that assesses an output model's expected predictive performance simply on data drawn from the same distribution as the training dataset (*e.g.,* a randomly held-out subset of the training dataset), the standard ML pipeline lacks any means to quantify the effectiveness of the trained models beyond what is captured by commonly-used metrics such as the F1-score. In effect, we argue that NetAI in its current form has contributed to a "dumbing down" of networking research as it has promoted a blind belief in the high-performant black-box models it considers.

We are not alone in criticizing the standard ML pipeline, its widespread and largely uncontested use across differ-

ent application domains of ML, and its overly pragmatic approach to evaluating the resulting trained models solely based on their effect (*i.e.,* "they work!"). For example, there is a growing body of work in the ML literature that is concerned with the surprisingly poor reported performance of many of the trained models that result from an application of the standard ML pipeline as soon as they get deployed in real-world environments [3, 4]. In fact, many of these works identify the fact that modern ML workflows such as the standard ML pipeline tend to be *underspecified* (*i.e.,* return many distinct models with equivalently strong test performance) as a key reason for why the resulting trained models do not generalize (*i.e.,* fail to perform as expected in deployment). Because of an evaluation procedure that relies on held-out data that have the same distribution as the training data, the standard ML pipeline has been shown to be especially prone to this underspecification problem, resulting in the observed poor model behavior in practice.

3. UNDERSTANDING CAUSE VS. EFFECT

As more of the failures and limitations of modern ML workflows such as the standard ML pipeline come to light, it is arguably justified to describe adhering to these workflows as being akin to "explorimentation" [1]. In fact, in the context of NetAI, we agree with the basic sentiment expressed in [1] that *"while appropriate for the early stages of research to inform and guide the formulation of a plausible hypothesis, [the standard ML pipeline] does not constitute sufficient progress to term the effort scientific."* Moreover, as NetAI is trying to overcome the general reluctance of network operators to deploy its ML-based solutions in their production networks, the standard ML pipeline's pragmatic "it works!" approach, typically quantified in terms of high F1-scores, to assessing its output by means of an iid evaluation procedure is no longer sufficient. In fact, to paraphrase [1], to ensure that network operators can begin to understand cause and not just effect of proposed ML-based solutions, *"it will be necessary to get past the stage of explorimentation and start employing tools of the scientific method."*

At the same time, we are not arguing for universally abondoning the use of ML workflows such as the standard ML pipeline and replacing them with "tools of the scientific method." For example, when employing ML-based solutions for low-stakes decision-making (*e.g.,* generic image classification, commercial recommendation systems, spam filtering), understanding how and why the underlying learning models make their decisions or knowing when they work or when they don't work is generally unnecessary or overkill — in such cases, for a black-box model to make a few wrong decisions is fully expected and tolerated, has little to no repercussions (*e.g.,* financial or reputation-wise), and can be fixed in future versions of the trained models. However, the situation is drastically different in cases where ML models are used for high-stakes decision-making (*e.g.,* predicting criminal recidivism risk, child welfare screening, medical treatment recommendation, self-driving vehicles) and where making a wrong decision or using an underspecified model can negatively impact the lives of people or the financial health or public reputation of companies. In such cases, innovative approaches that emphasize understanding "cause" over assuring "effect" which, after all, is the *raison d'être* of science, should be at the forefront of researchers' minds so they can successfully explain a trained model's decision-making process to end users who look for assurances that they can trust a proposed trained model.

In the NetAI domain, we say network operators "trust" a given ML-based solution if they are comfortable with relinquishing control to the model (see [4] and references therein). Given that network operators have remained reluctant to deploy trained models produced by the standard ML pipeline is evidence that they generally don't trust NetAI-based solutions. This applies in particular to trained models proposed for solving network security- or network performance-related problems where the consequences of a wrong decision can range from lost revenues, service contract terminations, customer dissatisfaction, and shutdown of business-critical services. As such, these models are clearly non-starters when it comes to engender trust in ML-based solutions among network operators. In contrast, "white-box" models such as decision trees promise to be ideal vehicles for convincing network operators that they can trust the models. These models not only describe in detail how and why every single decision is made, but domain experts can also examine them to find out when they work or don't work (and why not) and provide a means for scrutinizing the obtained model for indications of potential underspecification issues.

4. THE "OPENING UP" OF NETWORKING RESEARCH

To engender more trust in ML models, recent studies have argued for developing ML workflows that, instead of first creating black-box models and then trying to "explain" them, should generate white-box models such as decision trees that are inherently interpretable in the first place [2]. An attractive property of such workflows would be that they eliminate the need for any post-hoc explainability efforts because the models they output already reveal the underlying process by which they make their decisions and can therefore be directly checked and assessed by human domain experts, at least in theory. They also invite a direct comparison with the decisions domain experts would make when faced with the same data. However, as commented in [2], *"the belief that there is always a trade-off between accuracy and interpretability has led many researchers to forgo the attempt to produce an interpretable model. This problem is compounded by the fact that researchers are now trained in deep learning, but not in interpretable machine learning. Worse, toolkits of machine learning algorithms offer little in the way of useful interfaces for interpretable machine learning methods."*

Networking researchers contemplating developing ML-based solutions for their problems are therefore confronted with a serious dilemma. On the one hand, most modern ML pipelines, including the standard ML pipeline, focus almost exclusively on producing black-box models, but the use of such models in ML-based solutions that involve making high-stakes decisions is being increasingly criticized for the potentially tremendous harm they can inflict. The black-box models' inability to provide understanding in how and why they make their decisions also has the largely unintended consequence of contributing to a continued "dumbing down" of networking research. On the other hand, few, if any, ML pipelines exist today that have been explicitly designed to produce white-box models such as decision trees, even though their use in ML-based solutions that involve making high-stakes decisions is not

only preferred but recommended for the full transparency they provide for potentially life-altering decision-making. Moreover, the white-box models' ability to provide understanding in how and why they make their decisions makes them ideally suited for "opening up" networking research; that is, transforming networking research into a science by means of both a renewed focus on understanding "cause" and an intentional effort towards de-emphasizing "effect".

In an effort to resolve this dilemma, we recently developed and implemented TRUSTEE [4]. TRUSTEE defines a novel ML workflow that takes the trained black-box model (*i.e.,* model specification, training dataset) that results from an application of the standard ML pipeline as input and generates a white-box model in the form of a decision tree and an associated trust report as output. In synthesizing this decision tree, TRUSTEE strikes a balance between model fidelity (*i.e.,* accuracy of the decision tree with respect to the black-box model), model complexity (*i.e.,* the size of the decision tree and its explicitness and intelligibility), and model stability (*i.e.,* correctness, coverage, and robustness of the decision rules or branches of the decision tree). Using the decision tree that TRUSTEE extracts from the given black-box model, networking researchers can examine how or why the trained black-box model makes its decisions for a majority of data samples and can scrutinize it for indications of potential underspecification issues. Moreover, domain experts can use it to compare whether or not the black-box model makes the same decisions they would make when faced with the same data samples, and network operators can inspect TRUSTEE's output to gauge their trust in the given black-box model.

5. 2 STEPS FORWARD, 1 STEP BACK?

Early indications are that TRUSTEE has been a welcome and much-needed addition to the toolkits that researchers in the area of NetAI have relied on. As the use of ML in the networking domain continues to attract large numbers of researchers, TRUSTEE provides a concrete means for questioning some of the exhibited hubris and overconfidence by NetAI researchers, scrutinizing the soundness of NetAI-based solutions that have been reported in the existing literature, and performing some much-needed sanity checks on the myriad of proposed black-box models that have been trained for solving networking-specific problems. For example, by examining more than half a dozen of frequently cited and fully reproducible ML models from the existing networking literature, all of which are the results of using the standard ML pipeline, we found TRUSTEE to be especially good at refuting reported claims of "the model works!" and providing supporting evidence. In refuting these claims, TRUSTEE identified concrete instances of model underspecification issues, including trained models that leveraged shortcut learning strategies (akin to "cheating"), showed vulnerabilities to out-of-distribution samples (akin to "rote learning"), or exploited spurious correlations in the training data (akin to "lucky guesses"). The problematic nature of these findings argues for more caution with respect to the use of black-box models in the field of networking, suggests looking at developments in this area with a highly critical eye, and identifies common pitfalls or "blind spots" of proposed ML-based solutions that prevent operators from trusting them and deploying them in their production networks.

At the same time, while we agree with much of the reasoning in [2] where the author argues why interpretable black boxes should be avoided altogether in high-stakes decisions, our work with TRUSTEE caused us to take a more nuanced view with respect to explainable black-box models. For one, using the decision tree that TRUSTEE extracts from a given black-box model demystifies much of the decision-making process or "inner workings" of black-box models. In fact, this extracted decision tree becomes at once the main vehicle for domain experts to check if the given black-box model makes decisions in accordance with existing domain knowledge. An even more tantalizing application of a TRUSTEE-extracted decision tree is examining it with respect to the given black-box model's ability of teach the domain experts new decision-making strategies. Here, the term "teach" is meant in the sense of carefully inspecting the decision tree to see if it reveals legitimate strategies that the domain experts have been unaware of but upon painstaking examination recognize as meaningful and relevant decisions that deserve to be added to their existing domain knowledge.

6. CONCLUSION

We are presently not aware of any such examples in the NetAI domain where a given black-box model, via its TRUSTEE-extracted decision tree, teaches domain experts novel decision-making strategies. However, the likely existence of such examples suggests a natural "division of labor" in the NetAI domain between machines and humans that achieves the best of both worlds; i.e., leverages the raw computational power and algorithmic capabilities of ML to let machines do the grunt or "dirty" work (*i.e.,* sifting through training data, finding potentially useful patterns, and distilling them in a trained black-box model) and rely on the intelligence and inherently limited computing capabilities of humans to apply reasoning and logical thinking (*i.e.,* determining whether or not the detected patterns are meaningful and relevant for the problem at hand or point to possible underspecification issues with the trained black-box model). As this perspective explicitly argues for the need to keep human domain experts in the loop, it is counter to widely-held beliefs or common myths about the impact of increasingly autonomous technologies in general and NetAI-driven network automation in particular, namely that their wide-spread adoption will ultimately eliminate humans from the loop.In Part II of this NetAI Manifesto [5], we will revisit this perspective and argue why and how developing NetAI-based automation capabilities that work in practice will require keeping humans in the loop.

7. REFERENCES

[1] J. Z. Forde and M. Paganini. The Scientific Method in the Science of Machine Learning. *ICLR Debugging Machine Learning Models Workshop, 2019.*

[2] C. Rudin. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nat Mach Intell 1, 206–215, 2019.*

[3] A. D'Amor et. al. Underspecification Presents Challenges for Credibility in Modern Machine Learning. *Journal of Machine Learning Research 23, 1–61, 2022.*

[4] A. S. Jacobs et al.AI/ML for network security: The emperor has no clothes. *Proc. ACM CCS'22, 2022.*

[5] W. Willinger et al. A NetAI Manifesto (Part II): Less Hubris, More Humility. *Performance Evaluation Review, this issue, 2023.*

A NetAI Manifesto (Part II): Less Hubris, more Humility

Walter Willinger
NIKSUN, Inc.

Arpit Gupta
UCSB

Roman Beltiukov
UCSB

Wenbo Guo
Purdue Univ.

ABSTRACT

The application of the latest techniques from artificial intelligence (AI) and machine learning (ML) to improve and automate the decision-making required for solving real-world network security and performance problems (NetAI, for short) has generated great excitement among networking researchers. However, network operators have remained very reluctant when it comes to deploying NetAI-based solutions in their production networks. In Part I of this manifesto, we argue that to gain the operators' trust, researchers will have to pursue a more scientific approach towards NetAI than in the past that endeavors the development of explainable and generalizable learning models. In this paper, we go one step further and posit that this "opening up of NetAI research" will require that the largely self-assured hubris about NetAI gives way to a healthy dose humility. Rather than continuing to extol the virtues and "magic" of black-box models that largely obfuscate the critical role of the utilized data play in training these models, concerted research efforts will be needed to design NetAI-driven agents or systems that can be expected to perform well when deployed in production settings and are also required to exhibit strong robustness properties when faced with ambiguous situations and real-world uncertainties. We describe one such effort that is aimed at developing a new ML pipeline for generating trained models that strive to meet these expectations and requirements.

1. INTRODUCTION

In Part I of our NetAI manifesto [1], we criticize the use of the standard ML pipeline that is popular with NetAI researchers and discuss why relying on this widely-adopted ML workflow is fraught with problems that question the scientific foundations of the artifacts it produces. We argue for abandoning it altogether in favor of a new generation of ML pipelines that are capable of generating explainable ML models that can effectively be examined with respect to their generalizability and safety, describe an initial attempt at designing and implementing such a new ML pipeline, and comment on some exciting new opportunities that arise as result of this proposed paradigm shift in ML model development and evaluation.

In this Part II of the manifesto, we discuss how these proposed efforts towards "an opening up of NetAI research" relate to ongoing developments in the area of human-centered computing where the emergence of agents with increasingly autonomous capabilities are a major concern that dovetails with our calls for explainability and generalizability in everyday ML model development. Specifically, instead of further promoting the much-touted benefits of increasingly autonomous technologies, we elaborate on the urgent need to address the new risks and challenges these technologies pose and relate them to the popular view that the wide-spread adoption of NetAI-based autonomous capabilities will ultimately eliminate the need for human involvement.

In effect, we posit that responding to the new risks and challenges in a constructive manner demands a careful reappraisal that recognizes the existence of a natural "division of labor" between autonomous agents and humans that is counter to widely-held beliefs about the impact of increasingly autonomous technologies in general and NetAI-driven network automation in particular. To this end, we describe a novel ML pipeline that demonstrates the type of division of labor that can ensure a future where we can have the best of both worlds — the full benefits of autonomous capabilities without their possibly debilitating side effects such as "future automation surprises" [2].

2. NETWORK AUTOMATION AND AUTONOMOUS CAPABILITIES

The growing popularity of networked devices and applications imposes increasingly stringent security- and performance-related requirements on the underlying communication infrastructure. Satisfying these ever more demanding and complex requirements in an efficient and scalable manner with limited infrastructure resources and shrinking operational budgets poses significant challenges for today's network operators. A promising approach to address these challenges is to automate some of the real-time decision-making that satisfying these requirements necessitates. To this end, for more than a decade, networking researchers have been busy demonstrating the potential of NetAI, have developed NetAI-based solutions aimed at supporting network automation, and have been envisioning a future where NetAI will be critical for realizing the vision of "self-driving networks." Many of these efforts are, however, based on widely-held beliefs about the impact of increasingly autonomous technologies in general and NetAI-driven network automation in particular, namely that these developments will ensure that human involvement in the decision-making processes required for

operating and managing future networks can be reduced to the point where it will eventually become unnecessary.

Tellingly, these beliefs are collectively referred to as the "seven deadly myths" of autonomous systems in [3] where the authors systematically bust these "myths" of autonomy and provide reasons why each of them should be called out and cast aside. Although the authors of [3] take a broad view of autonomous systems and autonomous capabilities, their paper should be required reading for networking researchers, especially because the authors' observations and are directly applicable to and highly relevant for current efforts that focus on NetAI-driven network automation as a stepping stone towards realizing the future vision of self-driving networks. In the NetAI domain, the autonomous capabilities derive from running trained ML models in the data plane (e.g., on programmable switches) and relying on them to make real-time inference decisions. Here, each model can be viewed as an autonomous agent that has been designed with a specific networking task in mind (e.g., detecting the onset of an amplification-type DDoS attack), has been shown to be performant (according to some specified evaluation procedure), and the human operator feels comfortable relinquishing control to the model, thus automating a task that previously was performed by the operator.

In the context of NetAI, of particular interest is the first myth discussed in [3] that views "autonomy" as a unidimensional concept. Instead, the authors of [3] argue that it is more useful to describe autonomous agents at least in terms of the two dimensions referred to as *self-directedness* and *self-sufficiency*. Here, self-directedness is defined as the independence of an agent from its physical environment and reflects a notion of autonomy that is synonymous with independence from outside control. Self-sufficiency, on the other hand, is meant to capture the idea of self-generation of goals and reflects a view that equates autonomy with the capability of an agent to take care of itself. Slightly paraphrasing [3], a main motivation for autonomous capabilities is to reduce the burden on humans by increasing an agent's self-sufficiency to the point that it can be trusted to operate in a self-directed manner. However, when the self-sufficiency of the agent capabilities is seen as inadequate for performing the task the agent was designed for (e.g., in situations where the consequences of errors may be disastrous), it is common to limit the self-directedness of the agent, either by humans taking control manually or falling back to an automated control that is known to prevent the system from doing harm to itself or others through faulty actions (i.e., low self-directedness and low self-sufficiency).

When self-directedness is reduced to the point where the agent is prevented from fully exercising its capabilities (i.e., low self-directedness, high self-sufficiency), the result is an under-reliance on the technology — although the agent may be sufficiently competent to perform a set of actions in the current situation, human-imposed manual controls or policies may prevent it from doing so. The flip side of this aspect is over-trust (i.e., high self-directedness, low self-sufficiency) where an agent is allowed to operate too freely in situations that outstrips its capabilities. The challenge then faced by designers of autonomous agents or systems capabilities is striving to maintain an effective balance between self-directedness and self-sufficiency which in turn imposes the additional challenge on the designers to make the agent or system understandable. In NetAI parlance, these challenges are all-too-familiar. Making agents understandable is synonymous with developing explainable ML models, and model explainability is paramount for assessing model generalizability (i.e., assessing when the model works and doesn't wok (and why not)) and model safety (i.e., identify and quantify harmful and unintended model behavior).

3. AUTONOMOUS CAPABILITIES: RISKS AND CHALLENGES

As discussed in [3], like the mentioned first myth, most of the seven deadly myths or beliefs exist because they ignore or downplay in one way or another the new challenges and risks that materialize with increasingly autonomous capabilities and often take the form of "surprises" or "unintended consequences" that can reduce or even wipe out apparent benefits that may result from increased autonomy. The type of new challenges and risks is highlighted in [4] and has been termed "Doyle's catch" in [5]. It states that

"Computer-based simulations and rapid prototyping tools are now broadly available and powerful enough that it is relatively easy to demonstrate almost anything, provided that conditions are made sufficiently idealized. However, the real world is typically far from idealized, and thus a system must have enough robustness in order to close the gap between demonstration and the real world."

Although not expressed in NetAI language, we recognize Doyle's catch to be yet another formulation of the generalizability problem in ML — the failure of black-box models trained in the confines of an idealized setting (e.g., simple testbed) to maintain their performance when used in the real world (e.g., an actual production network). Thus, whether it is designing autonomous agents that are not prone to "surprises" or "unintended consequences", or developing autonomous capabilities without falling into Doyle's catch, or generating generalizable ML models, the technical challenge faced by researchers interested in developing ML-based solutions for networking problems is to define new ML pipelines that output trained models that are capable of "closing the gap between the demonstration and the real thing".

Since a majority of trained models that have been developed to date in the different application domains of ML are the result of applications of the standard ML pipeline, they are neither able to address nor resolve this challenging task. For one, since the output of the standard ML pipeline are in general black-box models, they provide little to no insights into the models' inner workings and instead continue to bolster the popular view that ML models are able to perform some "magic". Importantly, being black-box in nature, they are by and large unable to yield useful information for researchers interested in ascertaining the models' ability to generalize and there are currently no readily available ML pipelines that facilitate both the identification and remediation of underspecification issues in trained black-box models, a critical step in assessing the models' generalizability. Moreover, using the standard ML pipeline to obtain trained models has the effect of obfuscating the nature of the training data; that is, intentionally or unintentionally blurring critical information about the what, how and who of the collected data and thus about the data's quality.

Specifically, this data quality issue can be attributed to

two factors. First, many publicly available datasets are unrealistic in the sense that they have been collected from environments that have little to nothing in common with the "real thing" (i.e., the model's target environment). Second, existing data-collection efforts are fragmented; that is, they only apply to a specific learning problem and/or network environment. In particular, how to extend them to collect representative (labeled) data for a new learning problem and/or from a different target environment is a largely unresolved problem. Together, the obfuscating effect that the use of standard ML pipeline has with respect to the data employed for model training and the fact that the root causes of most model underspecification issues can be traced to problems with the quality of the training data [7] severely limit the options that researchers have to tackle the generalizability problem in ML. On the one hand, these issues highlight how the prolonged and widespread use of the standard ML pipeline in the different application domains of ML has resulted in a self-assured hubris among researchers in general and NetAI researchers in particular about the autonomous capabilities of ML-based agents. On the other hand, they also argue that in the face of the complexities and uncertainties experienced in the real world, some amount of humility is required to realize and accept that designing ML-based agents that are explainable and can be assessed with respect to their generalizability and safety cannot be accomplished by means of established methods such as the standard ML pipeline but demands implementing new ML workflows or pipelines that are radical departures from how ML models have been developed to date.

4. TREATING TRAINING DATASETS AS FIRST-CLASS CITIZENS

In the NetAI domain, the described tension between hubris and humility when researchers are faced with designing ML-based solutions for networking problems is greatly complicated by the fact that, in general, collecting data from the "real thing" to train ML models is, for privacy-related or other reasons, often not possible. In fact, the question of how to develop ML models that maintain their excellent performance even in the "unseen data" case (i.e., without an ability to collect data from "the real thing") while exhibiting the required balance between self-directedness (being robust to the uncertainties in the environment) and self-sufficiency (being able to perform safely despite the inherent fragilities that the complexity of autonomous capabilities entails) has largely stymied researchers in the past but deserves their full attention going forward.

In an initial attempt to resolving this taunting challenge, we recently incorporated TRUSTEE [7], our latest ML pipeline for developing explainable ML models into the design of a radically new closed-loop ML workflow that we call NETUNICORN [6]. NETUNICORN highlights two innovative and original concepts. First, it leverages a novel data-collection platform that enables the collection of different datasets for any given learning problem from one or more physical or virtual network infrastructures, accurately emulating different target environments with high fidelity. Second, NETUNICORN uses TRUSTEE-generated feedback about the latest trained model to iteratively collect new training datasets from some flexibly configurable idealized environment such that the models trained with these new datasets exhibit improved generalizability and have a better chance to maintain their good performance in the "real thing" (i.e., in the actual production network where collecting training data is ruled out). The premise is that the models that NETUNICORN outputs will instill greater trust among network operators for production deployments, thereby driving widespread adoption of ML in the field of networking in the future.

5. CONCLUSION

The novelty of NETUNICORN prevents us from reporting initial experiences from researchers who recognize the need for new ML pipelines that strive for outputting generalizable ML models. However, based on our own experience to date that admittedly encompasses only a small sample of different learning tasks and different target environments, the ML artifacts that result from applying our new closed-loop ML pipeline can be shown to be performant and to exhibit improved generalizability. However, only time will tell whether abandoning the standard ML pipeline with its deliberate tendency to rely on the "magic" of black-box models and implicit attempts at obfuscating the critical role of the underlying training data and replacing it with radically different ML pipelines such as NETUNICORN will have the intended consequences — the routine development of ML models that are both explainable and generalizable and where the role that the utilized data plays with respect to model training can be assessed explicitly. However, we posit that it is by developing new ML pipelines such as TRUSTEE and NETUNICORN that the NetAI domain can pave the way towards a future where ML models will be both recognized as a means for scientific discovery and appreciated for being of inherently practical value for achieving the autonomous capabilities required by ongoing efforts that see NetAI-based network automation as a stepping stone towards realizing the vision of self-driving networks.

6. REFERENCES

[1] W. Willinger et al.A NetAI Manifest (Part I): Less Explorimentation, More Science. *Performonce Evaluation Review, this issue (2023).*

[2] D. D. Woods. Automation Surprises. *In: Joint Cognitive Systems: Patterns in Cognitive Systems Engineering, 113–142, Taylor & Francis, 2006.*

[3] J. M. Bradshaw et al.The Seven Deadly Myths of 'Autonomous Systems'. *IEEE Intelligent Systems 28(3), 2–8, 2013.*

[4] D. L. Alderson and J. C. Doyle. Contrasting views of complexity and their implications for network-centric infrastructures. *IEEE SMC-Part A, 40(4), 839–852 (2010).*

[5] D. D. Woods. The Risks of Autonomy: Doyle's Catch. *Journal of Cognitive Engineering and Decision Making, 10(2), 131–133 (2016).*

[6] R. Beltiukov, W. Guo, A. Gupta, and W. Willinger. In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems. *https://arxiv.org/abs/2306.08853 (2023).*

[7] A. S. Jacobs et al.AI/ML for network security: The emperor has no clothes. *Proc. ACM CCS'22 (2022).*