HINT: Supporting Congestion Control Decisions with P4-driven In-Band Network Telemetry

HINT: Supporting Congestion Control Decisions with P4-driven In-Band Network Telemetry

Alessio Sacco * Antonino Angi * Flavio Esposito ‡ Guido Marchetto *
* Department of Control and Computer Engineering, Politecnico di Torino, Italy
‡ Department of Computer Science, Saint Louis University, USA
Invited Paper

***Abstract*—Years of research on congestion controls have highlighted how end-to-end and in-network protocols might perform poorly in some contexts. Recent advances in data plane network programmability could also bring advantages in transport protocols, enabling mining and processing in-network congestion signals. However, the new machine learning-based congestion control class has only partially used data from the network, favoring a more sophisticated model design but neglecting possibly precious pieces of data. In this paper, we present HINT, an in-band network telemetry architecture designed to provide insights into network congestion to the end-host TCP algorithm during the learning process. In particular, the key idea is to adapt switches' behavior via P4 and instruct them to insert simple device information, such as processing delay and queue occupancy, directly into transferred packets. Initial experimental results show that this approach comes with a little network overhead but can improve the visibility and, consequently, the accuracy of TCP decisions of the end-host. At the same time, the programmability of both switches and hosts also enables customization of the default behavior as the user's needs change.**

***Index Terms*—In-Band Network Telemetry, Congestion Control, P4**

I. Introduction

For the computer network to continue to thrive, it is necessary that congestion control mechanisms remain effective as the network evolves. It is not surprising, then, that several TCP variations have been proposed, given the increasing demand for ultra-low latency, high bandwidth, and network stability. There is a recent attempt to let the TCP learn the best congestion window size (*cwnd*) updates via learning-based schemes. A specific class of Machine Learning (ML), namely Reinforcement Learning (RL), has been proven to bring advantages to TCP flow and congestion control, having the ability to learn best updates based on collected experience and adapt them to the network environment [1]–[4]. Learning-based algorithms have shown great potential in automatically adapting to various network conditions, saving the engineering effort of manually tuning for unseen network conditions. It has been shown how end-to-end (e2e) TCP-based congestion control algorithms cannot detect if an increase in e2e delay measurements is due to another competing flow, stochastic packet losses, a route change, or the client delaying the acknowledgment [3], [5]. As such inefficiencies could lead to performance degradation, using network data may help detect actual network congestion.

At the same time, numerous recent In-band Network Telemetry (INT)-based network measurement solutions have been proposed to efficiently collect network data. They can measure, for example, one-way delay [6], tail latency [7], available bandwidth [8], queue depth [9]. Many advanced network management schema based on INT has also significantly improved network efficiency, including traditional congestion control [10], routing decisions [11], anomaly detection [12], and path tracing [7]. However, despite the improvements, none of the newly proposed TCP modifications can fully integrate network signals into the client's intelligence.

To this end, and to provide timely network feedback to the TCP logic, in this paper, we present HINT, a new solution based on incorporating in-band network telemetry into the newly RL-based mechanisms of TCP. In particular, this work aims to analyze how it can support and improve TCP protocol decisions by collecting network statistics on devices and then processing them through the RL algorithm on hosts. In fact, our solution is constituted of two main components: (1) a modified instance of TCP that runs on the end hosts and (2) intelligent network devices to empower network telemetry. We use the P4 [13] language for device programming, given the vast gamma of metrics available and the short processing time that the PISA architecture provides. Our solution overcomes some limitations of standard protocols by integrating the intelligence of ML algorithms with incoming fresh network information. We feed P4 collected metrics to the RL model, i.e., switch processing time and queue occupancy, giving the host capabilities to understand different situations and react adequately to the network's congestion level.

Our results have shown that P4-enabled switches effectively collect a heterogeneous set of metrics, and we experimented with how multiple packet header formats can be used. Moreover, we showed that when such metrics are used by a target TCP algorithm, i.e., Owl [3], its decision-making ability improves, leading to better results in terms of both throughput and RTT. Moreover, we quantified the advantages of HINT compared to other TCP congestion control protocols for various network loads and sizes.

The remainder of the paper is structured as follows. Section II presents current INT mechanisms and TCP congestion control algorithms. In Section III we describe the main functionalities offered by HINT, while Section IV explains how it can be used for TCP congestion control. Performance is

quantified in Section V, and Section VI concludes the paper.

II. Related Work

In-band Network Telemetry. In-band network telemetry (INT) is an emerging modality of network telemetry, that collects the network status and parameters, such as the load on the links, queuing latency, or queue size, by inserting switching nodes' metadata directly into packets [14], [15]. INT-path [9], for example, couples the INT probe with the source routing label stack to accommodate the user-specified monitoring path. This solution, as many other recent ones, are built upon the P4 language for data collection and consumption because of the facility provided by the language to operate by adapting to various packet formats and protocols [14], [16]. Given its success in this area, P4.org defined the INT data plane specification and formalized INT notations and specifications for a general architectural model that can fit many applications [17]. It also defined the set of information to carry in the INT header, and INT-MD is a widely used standard format that allows inserting information and instructions in the packets. Recent approaches focus on a trade-off between completeness of the information and switches/bytes overhead to reduce the load on the network and reduce the amount of calculation required to obtain data from the telemetry platform layer [7], [18].

Recent network-assisted CCs. Although hosts can use the rich network status information provided by in-band network telemetry for a broad range of operations, in this paper we specifically focus on new congestion avoidance and control protocols. Several protocols leverage the Explicit Congestion Notification (ECN) as insightful feedback to end hosts, e.g., the well-known DCTCP [19], or XCP [20], that modify switches behavior to feedback rates to end-hosts. NATCP [5], HPCC [10], ABC [21] and Swift [22] are recent approaches that leverage switches (or a centralized entity for NATCP) to send information about bottleneck links. While ABC [21] improves on ECN by sending accelerate and brake signals instead of merely random early drop signals, High Precision Congestion Control (HPCC) [10], instead, is a data center network load balancing scheme based on INT to obtain precise link load information.

Compared to all other solutions, HINT appears to be the less invasive as possible, demanding a slight modification of the network nodes that make it portable in a variety of contexts. Moreover, it is the first INT for TCP integration that addresses the new design and challenges posed by the new trend of TCP congestion control: RL-based protocols.

III. System Design

We summarize in Fig. 1 the main components of our solution. In particular, we can observe that HINT revolves around modifying the traditional behavior of network switches and introducing a custom header in the packet.

HINT switch. The network switches are INT-capable network devices that participate in the INT data plane by inserting, adding, removing or processing data from INT headers in packets. Examples of INT Nodes may include devices such as routers and switches, which we envision to be instructed using P4 language [13]. These nodes need to insert their device-internal state into incoming packets and forward them to the destination defined in the packet.

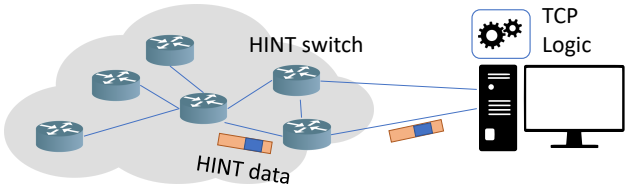


Fig. 1: HINT overview. Network switches insert telemetry data as additional header, then used by the TCP end-host program to improve learning logic.

HINT header. The INT Source Node does not need to mark any telemetry packet, and only switches have to parse, read, and write the network information that would be received by the INT receiver. The HINT network feedback inserted by each switch is composed of the following metrics: *(i) switch ID*, containing the id number of each switched crossed (13 bits), *(ii) queue occupancy*, indicating the number of packets queued in each switch buffer (13 bits), *(iii) hop latency*, the switch processing time (6 bits), computed as follows:

$$hop\ latency = packet\_egress\_time - packet\_ingress\_time. \quad (1)$$

We designed this set of information as it can be easily obtained in most switches. For example, P4 language provides these metrics in the metadata fields $egress\_global\_timestamp$ and $ingress\_global\_timestamp$. Queue length and hop latency give a specific overview of the network's congestion level while the switch ID helps detect the less-performing switch. When these network signals indicate that congestion occurs, it is easy to apply mitigation strategies to reduce sending rate. Lastly, as can be seen, the feedback referring to each switch is 32 bits, making the telemetry data compatible with standards [17] and the variety of protocols considered during design and prototype development (Section III-A).

A. Exploiting Programmability for Protocol Format and Header Stack

We set the network nodes to work in two alternative modes: *(i) push*, *(ii) update*. In the first approach, switch information, which is viewed as a group of 32-bits, is stacked into the INT header, allowing the receiver to process data with more operations, not only aggregated information as finding maximum or minimum. For example, INT Sink, i.e., the INT endpoint and data collector [17], can digest the information and perform latency and INT packet loss analysis before sending aggregate information to the final host or controller. Although we do not consider the Sink in our architecture, letting the host receive the metadata and process them (since the host already requires socket modification) is a viable extension to further operations (e.g., computing latency distribution in the path).

In the second approach, the HINT data of the header are updated, reducing the network overhead. As this paper mainly focuses on congestion control, we design this option as the default and instruct switches to maintain the max value of the queue depth and processing time in the path, and the switch ID of the nodes that last updated the header. Having insight into the worst network condition would help the host to solve the network congestion [3], [20], [23].

Although we set the default version of HINT to work with IPv4 options, in what follows we describe in detail three other possible protocols which our solution is compatible with, highlighting the advantages and disadvantages of each of them. Aside from the ones suggested in [17], we envision these valid alternatives: *(i) IPv4 Options*, *(ii) Multiprotocol Label Switching (MPLS)*, *(iii) Big Packet Protocol (BPP)*. *(i) IPv4 Options* are extra 32-bit words that can hold options about packet treatment inside traditional IPv4 protocol. This piece of header carries the information defined by HINT in the selected format of Section III. IP Options result in a less invasive format and are the most used approach for INT solutions. As such, we set it as default but extended the final product with some alternatives that can be used as the business demands change. *(ii) MPLS* is a well-known routing technique based on the key idea that packet-forwarding decisions are made on the contents of labels assigned to data packets. Labels prefixed by MPLS constitute the MPLS header and form a label stack. Our MPLS-compliant version of HINT would use this field to carry the network telemetry information. The popularity of this protocol would favor considerable interoperability on both switches and hosts, at the cost of an additional packet header that, compared to IPv4 options, demands more extra bytes. *(iii) BPP* is a newly defined approach to customize packet-based networking behavior based on the introduction of a piece of BPP Collateral into packets as an additional packet header [24]. This block of data, traditionally used to provide guidance to intermediate network devices on how to process those packets, is used in our BPP-compliant version to acquire switch details about congestion. We can easily observe how this option, despite implying a larger amount of additional bytes (almost 30 bits) compared to the IPv4 approach, can be easily extended to offer an accurate quality of service guarantees and to facilitate other traffic engineering operations that are simply not possible in the IPv4 header.

Our network programming framework based on data plane programmability in general, and P4 in particular, can indeed be used to support these policies and easily adapt to applications with different demands without needing to deploy custom hardware in networking devices and without the intervention of sophisticated controllers.

IV. End-Hosts' TCP Integration

Although our architecture can fit other network problems (see Section VI), one scenario we specifically consider in this paper is the TCP Congestion Control Algorithm (CCA). As mentioned in Section II, the idea of using network information for the end-host TCP decisions is not new. Especially in challenged network scenarios such as cellular access links, it may be convenient that end-hosts get help from the network itself [3], [5]. These approaches are based on the periodic collection of network information about bandwidth and delay, sent to the host as digested feedback. This feedback is used to set *cwnd* and pacing rate, and send data accordingly. However, the new wave of CCAs is moving toward autonomous learning techniques, such as Machine Learning (ML) and Reinforcement Learning (RL), to define the sending rate [1]–[3]. On the one hand, this new class of solutions enables new possibilities in the decision process by incorporating many heterogeneous signals. On the other hand, it poses new challenges in the design to reach an accurate but feasible implementation.

In this regard, we considered a particular TCP solution to prove how it can benefit from HINT. We integrated HINT with a learning-based CCA, namely Owl [3] since it has already been designed to receive some information from the partially known network. Using an RL formulation, Owl learns the best actions based on collected experience via interacting with the training environments. Thus, it can adapt itself to various conditions without the need to be tuned or manually engineered for every unseen scenario. Moreover, it is built upon the Cubic implementation in order to speed up the learning process and improve generalizability.

Every reinforcement learning (RL) problem involves a decision-maker (agent) attempting to learn the behavior of a dynamic system through repeated interactions [25]. Specifically, an agent receives the current state and the reward from the dynamic system at each iteration and outputs an action that maximizes a specified objective. As a result, the agent receives state and reward from the system, while the only input the system receives from the agent is the action. The success of the agent's action decisions are indicated by a reward value, and the agent learns which actions to choose to provide the highest cumulative reward over time, i.e., the long-term revenue.

In Q-Learning [26] the value of executing an action from a given state is estimated in a Q-value, stored in a Q-Table. However, to deal with the large state and action spaces, such a Q-Table is approximated through neural networks, originating the deep reinforcement learning (DRL) and, specifically when applied to Q-Learning, deep Q-learning.

The majority of DRL-based CCA aims to continuously select the next action, i.e., congestion window size, that maximizes the cumulative reward. In the case of Owl, the *state space* is composed of 10 features, namely: (i) congestion window size (*cwnd*), (ii) round trip time (RTT), (iii) RTT variation between two consecutive samples, (iv) maximum segment size (MSS), (v) number of delivered packets, (vi) packets lost during a transport session, (vii) current packets in-flight, (viii) number of retransmissions, (ix) partial network congestion, (x) the percentage of known network. The *action* is the offset between the newly selected *cwnd* and the current one. In particular, the action set is composed of $\{-10, -3, -1, +0, +1, +3, +10\}$. The DRL model, by selecting one of these actions, learns how to make control decisions from experience and eliminates the need for necessary pre-

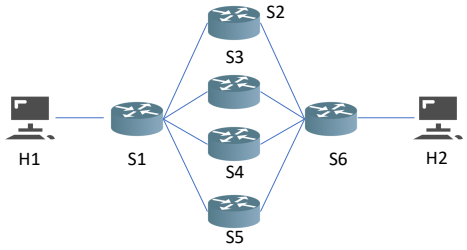


Fig. 2: Network topology used in the experimental evaluation.

coded rules to adapt to the variety of network environments. The *reward* function models the application-level goal of "high throughput and few losses." In particular, the reward R_i of sender i is a function of throughput of client i (λ_i), packet loss rate for i (p_i), as follows:

$$R_i = \lambda_i - \delta_i \lambda_i \left(\frac{1}{1 - p_i} \right),$$

where $p \in [0, 1)$ and δ is an adjustable coefficient determining the importance of the components. For further details on the DRL model, we point this reference for the reader [3].

We modified its instance in order to integrate it with our approach and make it able to accept information with this new format. In particular, the Deep Reinforcement Learning (DRL) model of Owl running on the host is thus fed with these three more metrics from the network. The neural network shape of the DRL is increased, and the original partial network information is replaced by these three values. In this use case, we only consider the *update* version of HINT where the header contains the max values of queue occupancy and hop latency. This operation limits the number of network feedback always to three and makes HINT portable over larger networks, as the header is limited in its growth. We thus intervened in the kernel implementation to extract the data and pass it to the machine learning model.

V. Evaluation Results

In this section, we illustrate our evaluation results showing the advantages of our solution in terms of network efficiency and TCP integration.

Experimental settings. We deploy HINT over Mininet, a network emulator that enables the reproduction of arbitrary virtual networks for rapid prototyping.Supporting the software-defined networking (SDN) approach, it can also create switches that are P4 compliant thanks to Behavioral Model Version 2 (bmv2), which enables translating P4 code into C++11 software switches' packet-processing operations. The P4 compiler is built upon standard version *p4-16*. The topology used for testing is shown in Figure 2, where all links have 100 Mbps of bandwidth, and packets crossing the network are routed using Equal Cost Multi-Path (ECMP) strategy. $H1$ acts as a client transmitting data using $iperf3$, and $H2$ is the server of the communication, and if not otherwise specified, we create other traffic to have all the links 40% utilized.

Packet formats adaptation. First, we compute the RTT and Flow Completion Time (FCT) (as suggested in [27]) of the transmission for different durations of the flow, reporting results in Figure 3 after 15 trials for each setting. We compare the header formats compatible with HINT, i.e., IPv4, MPLS, BPP, and the standardized INT-MD metadata header [17], which has been recently used in other articles, e.g., [28].

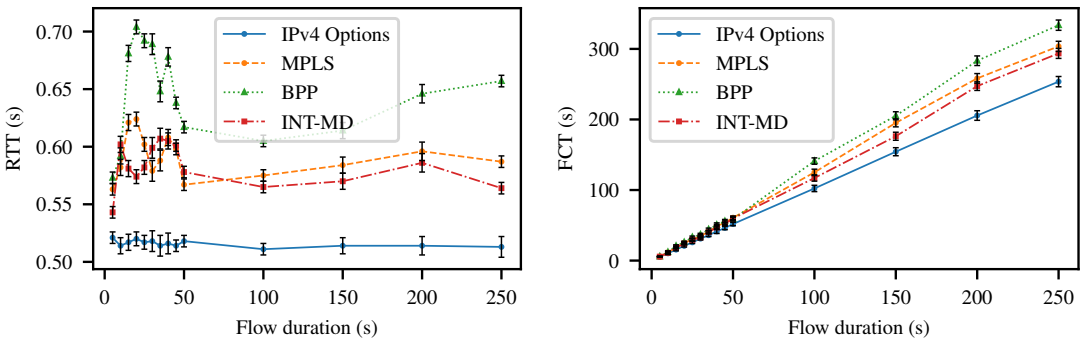
We can easily observe how the IPv4 options-based version is able to reduce the average RTT and FCT for all possible durations of communication. The fewer bytes to be transmitted and processed, along with the consequently more efficient P4 implementation, can significantly reduce the impact of INT. Moreover, we can also note that none of these protocols increase the impact of network congestion, having a limited impact on both RTT and FCT. Therefore, we can conclude that all options are valid, but the IPv4-compliant is less invasive from the network and switch side. As such, we set this option as the default version of HINT and use it in the following.

TCP integration evaluation. After observing efficacy in design and P4 implementation, we now quantify the goodness of metrics selected in the case of TCP CCA. We modified Owl source code to assess the advantages of this new mechanism to carry in-band information and deployed the new prototype in the same network topology. We compare the traditional Owl and this new version after training both models for 1 hour.

Figure 4 shows the evolution of (a) RTT and (b) throughput for the two Owl versions during a 2 minutes transmission. We can clearly observe how HINT is optimally integrated into CCA and allows the latency reduction and an increase of the throughput jointly. In particular, the INT-based approach can deliver network information to the host in a faster way compared to the original interval-based approach. While traditional Owl uses network information every time interval (1 second in our implementation as suggested by the authors), the decision logic uses periodic feedback that is unaligned with the packet transmission. Instead, having the feedback directly in the packet allows associating the statistics to the packets, resulting in a more accurate learning and more responsive *cwnd* adjustment.

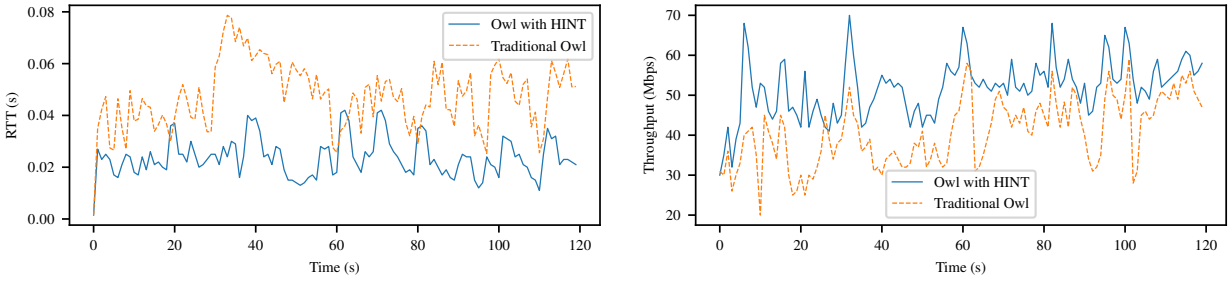
For example, nearly from second 30 to 60, the original Owl algorithm is slower in detecting congestion and then reacting to this event (Figure 4a). The modification based on INT, on the contrary, helps towards the reactive response of the host, keeping the RTT always limited. The same occurs for throughput (Figure 4b), interval 40-60 seconds, where throughput reduction because of high network congestion only partially affects the transmission, which continues without excessive throughput downgrade. The feedback received by the network helps the host to follow the congestion in the path and properly adapt the *cwnd* mechanism.

Network utilization impact. Clearly, throughput and RTT depend on the network utilization, and we measure the effect of traffic in Fig. 5a-b. We quantify the advantages oh HINT compared to other CCAs, namely: *(i)* Cubic [29], a traditional protocol, which is the default on Unix systems, *(ii)* Aurora [1], an RL-based CCA, which learns without network feedback, *(iii)* Owl, the reference CCA, which lacks an INT mechanism, *(iv)* ABC [21], a CCA employing in-network control, i.e., the



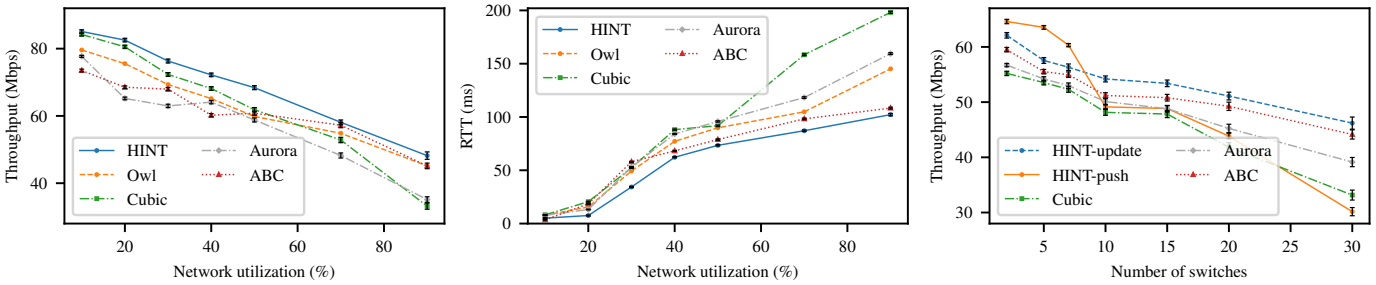
(a) (b)

Fig. 3: (a) RTT evolution and (b) FCT evolution at increasing volume of sent traffic. Our IPv4-based version can reduce network delay.



(a) (b)

Fig. 4: (a) RTT evolution and (b) throughput of 120 seconds transmission. The Owl version based on HINT leads to minor RTT and higher throughput compared to the traditional version of Owl.



(a) (b) (c)

Fig. 5: **Network utilization.** (a) Throughput and (b) RTT evolution for increasing network utilization. **Larger topology.** (c) Throughput performance over different network topologies and increasing number of informing switches. .

network devices send congestion information and command to accelerate or break. We define network utilization as the ratio between consumed bandwidth and total capacity in the environment. We can observe that other in-network congestion control mechanisms, e.g., ABC and Owl, are ineffective with low network utilization, to the point that a simple CCA as Cubic leads to a higher throughput. The advantages of a network feedback are more visible when network utilization touches 70%, both in terms of throughput and RTT. At this point, Cubic and Aurora can not properly handle network congestion. We can however observe that HINT consistently increases the throughput and lowers the latency compared to all alternatives, given the limited overhead introduced in the network.

Network size impact. We now compare the two versions of HINT against a few representative protocols when increasing the number of switches over randomly generated topologies, i.e., links are randomly generated while we fix the network size. The link capacity is also uniformly distributed at random between 50 and 100 Mbps. We measure the perceived throughput when our solution runs the *update* version of telemetry and *push* version, reporting results in Fig. 5c. It can be observed that the completeness of knowledge brought by *HINT-push* is beneficial when the number of switches is limited, around 10 switches in the topology. When the network size grows, *HINT-update* appears as the preferred option that is also able to outperform alternatives. This result is extremely important to understand how the HINT information helps the TCP CCA to

maximize the throughput and that both versions of our solution are well-designed, but *HINT-update* must be used for larger networks.

VI. Conclusion

This paper presented HINT, a novel solution that, using In-Band Network Telemetry, delivers valuable and timely information to the end-hosts in order to handle congestion. In particular, we studied how different protocols, i.e., IP Options, BPP, and MPLS, can gather the status of traversed P4 switches and inform the TCP module of hosts to properly adjust sending rate. Results in our P4 implementation confirmed how a limited set of information could effectively obtain an indicative network congestion level and help TCP process on hosts. We plan to extend this work in the near future in several directions: more realistic workloads and alternative TCP protocols.

In the future, we plan on extending HINT to other TCP protocol. While this paper presented a specific set of network metrics and a particular TCP use case, this system can be easily extended to work with more packet formats or with a different set of statistics, and can cooperate with other transport protocols and congestion control algorithms, e.g., NATCP, Swift, or DCTPC, to name a few.

Acknowledgment

This work has been partially supported by the Comcast Innovation Fund and the NSF awards 1647084, 1836906, and 2201536.

References

[1] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 3050–3059.

[2] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '20)*. Association for Computing Machinery, 2020, p. 632–647.

[3] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Owl: congestion control with partially invisible networks via reinforcement learning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[4] Y. Ma, H. Tian, X. Liao, J. Zhang, W. Wang, K. Chen, and X. Jin, "Multi-objective congestion control," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 218–235.

[5] S. Abbasloo, Y. Xu, H. J. Chao, H. Shi, U. C. Kozat, and Y. Ye, "Toward optimal performance with network assisted TCP at mobile edge," in *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. USENIX, 2019.

[6] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, and J. Wu, "Netview: Towards on-demand network-wide telemetry in the data center," *Computer Networks*, vol. 180, p. 107386, 2020.

[7] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '20)*. Association for Computing Machinery, 2020, pp. 662–680.

[8] N. S. Kagami, R. I. T. da Costa Filho, and L. P. Gaspary, "Capest: Offloading network capacity and available bandwidth estimation to programmable data planes," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 175–189, 2019.

[9] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, and Y. Liu, "Int-path: Towards optimal path planning for in-band network-wide telemetry," in *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 2019, pp. 487–495.

[10] Y. Li, R. Miao, H. H. Liu *et al.*, "HPCC: High precision congestion control," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, 2019, pp. 44–58.

[11] A. Karaagac, E. De Poorter, and J. Hoebeke, "Alternate marking-based network telemetry for industrial WSNs," in *16th IEEE International Conference on Factory Communication Systems (WFCS)*. IEEE, 2020, pp. 1–8.

[12] S. Nam, J. Lim, J.-H. Yoo, and J. W.-K. Hong, "Network anomaly detection based on in-band network telemetry with rnn," in *2020 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. IEEE, 2020, pp. 1–4.

[13] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[14] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM Conference Posters and Demos*, vol. 15, 2015.

[15] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu *et al.*, "Packet-level telemetry in large datacenter networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. Association for Computing Machinery, 2015, pp. 479–491.

[16] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: A survey," *Computer Networks*, vol. 186, p. 107763, 2021.

[17] P. A. W. Group. (2020) In-band network telemetry (int) dataplane specification v2.1. [Online]. Available: https://github.com/p4lang/p4-applications/ blob/master/docs

[18] H. Namkung, Z. Liu, D. Kim, V. Sekar, and P. Steenkiste, "SketchLib: Enabling efficient sketch-based monitoring on programmable switches," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Usenix, 2022, pp. 743–759.

[19] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '10)*. Association for Computing Machinery, 2010, p. 63–74.

[20] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '02)*. Association for Computing Machinery, 2002, pp. 89–102.

[21] P. Goyal, A. Agarwal, R. Netravali, M. Alizadeh, and H. Balakrishnan, "ABC: A simple explicit congestion controller for wireless networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX, 2020, pp. 353–372.

[22] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, and A. Vahdat, "Swift: Delay is simple and effective for congestion control in the datacenter," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '20)*. Association for Computing Machinery, 2020, p. 514–528.

[23] A. Sacco, F. Esposito, G. Marchetto, and P. Montuschi, "A self-learning strategy for task offloading in uav networks," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 4, pp. 4301–4311, 2022.

[24] R. Li, A. Clemm, U. Chunduri, L. Dong, and K. Makhijani, "A new framework and protocol for future networking applications," in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*. Association for Computing Machinery, 2018, pp. 21–26.

[25] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[27] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[28] S. Xie, G. Hu, C. Xing, J. Zu, and Y. Liu, "FINT: Flexible In-band Network Telemetry method for data center network," *Computer Networks*, p. 109232, 2022.

[29] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.