P4Flow: Monitoring Traffic Flows with Programmable Networks

Habib Mostafaei, Shafi Afridi

***Abstract*—Cloud providers perform flow monitoring to get insights from the network traffic flows, often using coarse-grained packet counters or packet probing. These approaches give partial information from ongoing flows or introduce significant overhead if the probe packets cross multiple hops with diverse delay and bandwidth to reach the traffic collector. Recently, In-band Network Telemetry (INT) offered by programmable networks, e.g., P4, has gained attention by providing fine-grained network telemetry. Current attempts on INT are inflexible in collecting telemetry for customs flows according to the desired interval. This letter proposes P4FLOW, a flow monitoring tool for cloud provider networks implemented on programmable data planes. P4FLOW allows the providers to monitor a set of desired flows according to their needs. It reduces at least 1.6x the overhead of telemetry packets compared with the existing approaches.**

***Index Terms*—Flow monitoring, Programmable networks, In-band Network Telemetry (INT).**

I. Introduction

The network infrastructure of the cloud providers often consists of thousands of switches/routers connected through the links [1], [2]. The services, e.g., video streaming or congestion control, running on these infrastructures generate a massive number of traffic flows. The providers need monitoring tools to get insights from ongoing network flows. SNMP, ping, or other legacy tools can provide valuable information on the status of flows. However, this can be improved by employing the recent advances in programmable networks such as P4 [3]. The provider can get insights from the packets/flows and the intermediate devices by using the In-band Network Telemetry (INT) [4] provided by P4. However, getting the flow statistics from each device adds significant overhead to the network. Therefore, the providers desire to collect the flow information from one of the devices in the path from a source to a destination according to the application's need.

Many techniques exist to collect flow statistics [5], [6], [7], [8], [9]. OpenTM [7] and FlowCover [8] try to minimize the cost of steering the telemetry information by selecting a node from a source to a destination in SDN-based networks. Nevertheless, these approaches add significant overhead to the network. The goal of INT-path [6] and P2INT [9] is to cover all the networks' links to collect the flow statistics in a P4-based network. The work in [10] concentrates on coordinating of probe packets to get insights from the network flows. However, these approaches focus on coordinating probe packets to collect the flow statistics without considering the overhead of steering probe packets to the collector.

In this letter, we propose a flow monitoring tool P4FLOW that uses simple additive weighting technique to find the best network devices to report the flow telemetry in a P4 network. It adds less overhead to the network while providing the flexibility to the users to poll the flow telemetry according to their desired intervals and flows. We summarize our contributions as follows.

- We propose a simple additive weighting-based technique to select a leader node to poll the flow statistics.
- The users of P4FLOW can adjust the interval of having flows telemetry information according to the applications' needs.
- We report that P4FLOW injects significantly fewer packets compared with the existing techniques and improves the flow completion time.

We make our code publicly available and our results fully reproducible[1].

II. Problem Formulation

We model the network as an undirected graph G=(P,E) in which P=$\{p_1, p_2, \dots, p_m\}$ is a set of P4 switches and E=$\{e_1, e_2, \dots, e_m\}$ is the set of edges among them. The network steers the traffic of n active flows $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$ where each flow f consists of a set of P4 switches from a source to a destination. We can report the flow information using each P4 switch along the path, but this solution adds additional overhead to the network due to injecting a large number of packets. Therefore, our goal is to select a P4 switch, i.e., a leader switch, along each flow f to report the flow information to the controller[2]. We call $\mathcal{L}_f$ as the leader switch for flow f. We assume that the first node in the graph is the controller of the network being in charge of collecting the flow information. To avoid possible congestion imposed by the flow telemetry packets, we aim to steer the traffic for telemetry packets via less congested links with high available bandwidth.

Delay. In delay-sensitive applications, the flow information should be forwarded to the controller with a short delay to take a suitable reaction to the network changes. In such scenarios, minimizing the delay between the leader switch and the controller is desirable. Let D_{ij} denotes the link latency from switch p_i to p_j. We denote the sum of link delays along the path from the leader switch $\mathcal{L}_f$ to the controller for flow f as $D(f)$. Therefore, our goal is to minimize the value of $D(f)$ for each flow in $\mathcal{F}$.

H. Mostafaei and S. Afridi are with the Department of Telecommunication Systems, Technische Universität Berlin, 10587 Berlin, Germany, e-mail: {habib, safridi}@inet.tu-berlin.de.

[1]https://github.com/mostafaei/P4Flow

[2]We use the general term of the controller in this work since in our implementation the P4 runtime plays the role of the controller.

Bandwidth. We desire to send the flow telemetry to the controller using the links with the highest available bandwidth. In that case, the link with the minimum available bandwidth specifies the available bandwidth. Let B_{ij} denotes the available bandwidth from switch p_i to p_j. We can compute the available bandwidth for flow f, i.e., $B(f)$, by taking the minimum available bandwidth of all links along the path from $\mathcal{L}_f$ to the controller. Herein, our goal is to maximize $B(f)$ for each flow in $\mathcal{F}$.

Objective functions. The main goal is to minimize the delay of steering the flow telemetry to the controller while obeying the bandwidth constraints of the links. We use a simple-additive weighting method [11] to transform the multi-objective selection problem into a single one. We define our objective function F as the weighted sum of normalized values for delay and bandwidth for each flow f as follows.

$$F(f) = w_d \times \frac{D(f) - D_{min}}{D_{max} - D_{min}} + w_b \times \frac{B(f) - B_{min}}{B_{max} - B_{min}} \quad (1)$$

where $w_d, w_b \geq 0$ and $w_d + w_b = 1$, are weights for link latency and the available bandwidth. $D_{min}(D_{max})$ and $B_{min}(B_{max})$ are the minimum (maximum) value for overall expected latency and available bandwidth. We normalize the values of link latency and available bandwidth to adjust the selection metrics. After normalization, each value varies in the range of $[0, 1]$, where the value of 0 corresponds to the worst value, i.e., $D(f) = D_{max}$ and $B(f) = B_{min}$, and the value of 1 corresponds to the best value, i.e., $D(f) = D_{min}$ and $B(f) = B_{max}$. Therefore, our objective is to minimize $F(f)$, mathematically,

$$\min_{\forall f \in \mathcal{F}} \quad F(f)$$

This model can be extended to include other parameters, e.g., the cost of sending the flow information, in selecting $\mathcal{L}_f$ for each flow f.

III. The Algorithm

In this section, we first present a greedy approach to select a switch in a flow to send the flow information. Then, we explain the P4 implementation of P4Flow in detail.

A. The Switch Selection Algorithm

The control plane of P4Flow takes the set of flows $\mathcal{F}$ and the coefficient for the link parameters, i.e., latency, and bandwidth, as the inputs. Then, it checks the value of the objective function in eq. 1 and selects the switch in the path with minimum value as the leader switch to send the flow information. Algorithm 1 presents the pseudo code of the leader selection algorithm in P4Flow.

The algorithm checks the possibility of each switch in the flow for being a leader switch to report the flow information. To do so, the leader selection algorithm checks its distance to the controller using the inter-node delays and their available bandwidth. There might be multiple paths from a switch to the controller, and we use the Dijkstra algorithm to select the best path. It is a single-source shortest path algorithm that works with non-negative edge weights. The running time of Dijkstra's algorithm is less than other algorithms such as Bellman-Ford's one for the same problem. Furthermore, the time complexity of Dijkstra's algorithm can be improved using the Fibonacci heap. The network administrators can adjust each link's cost according to the objective function in eq. 1 since it has a non-negative value. After selection of the best path, we compute the objective function of eq. 1 to check the suitability of the switch. We select the switch with a minimum value of F in the flow as the leader. Then, the algorithm adds this node to the list of leader switches $\mathcal{L}$. We repeat the same procedure for all the flows.

Algorithm 1: Leader node selection

```
input : Network graph $G$, links delay and bandwidth, flow list ($\mathcal{F}$)
output: A list of leader switches $\mathcal{L}$
Function findLeaderSwitch():
    $\mathcal{L} = \emptyset$ ;
    for each $f \in \mathcal{F}$ do
        t = ∞;
        for $p \in f$ do
            $F_p$ = apply eq. 1;
            if $F_p < t$ then
                $t = F_p$;
                $\mathcal{L}_f = p$;
            end
        end
        $\mathcal{L} = \mathcal{L} \bigcup \mathcal{L}_f$ ;
    end
    return $\mathcal{L}$;
End Function
```

B. P4 Implementation

P4Flow can monitor all the packets of all the flows in the network. However, this adds significant overhead to the network, which operators are reluctant to do. Therefore, P4Flow allows the operator to define a time interval for the flow telemetry. For example, if the flows are sensitive to the delay, the threshold value for the time interval can be chosen as a small value. While for other traffic flows, a large threshold value is desirable to reduce the overhead. This value can be tuned using the control plane rules. We use a P4 register to keep the threshold value for the interval. To do so, the network administrators can set the initial value for the threshold depending on the services offered by the cloud service providers. If the flows carry the traffic of the critical business transaction information, the threshold value for the time interval has to be small to identify the problem quickly [2]. For example, Akami [12] in 2017 report that each 100ms of network delay results in a 7% revenue reduction. For other flows, such as the flows that carry the email data, the time interval for the threshold can be set to a high value.

P4Flow consists of two main phases, namely, *tracking flows* and *collecting telemetry*. It performs flow tracking in the ingress control flow of each switch to specify where the packet has to be sent, i.e., to the next hop or the controller. Collecting flow telemetry is carried out in the egress control flow. We now explain the P4 implementation of our tool.

Tracking flows. To track the flows, we need to keep the state of each flow in the leader switches. Upon entering a packet of a flow to the ingress control flow, P4Flow performs a

```
register<bit<48>>(1) pkt_thd_reg;
register<bit<48>>(MAX_ENTRIES) timer_reg;
action do_clone_e2e(){
  clone3(CloneType.E2E,E2E_CLONE_SESSION_ID,meta);  }
action mark_packet(){
  meta.flag = 1;
  data=hdr.ipv4.srcAddr ++ hdr.ipv4.dstAddr ++ hdr.tcp.
     srcPort ++ hdr.tcp.dstPort ++ hdr.ipv4.protocol ++
     meta.last_timestamp ++ meta.packet_size;
}
table generate_clone{
  actions = {
    do_clone_e2e;
    NoAction; }
  default_action = NoAction();
}
apply {
    pkt_thd_reg.read(meta.pkt_thd, 0);
    time_t timer_cnt;
    time_t timer_tmp;
    time_t cur_time = standard_metadata.
     ingress_global_timestamp;
    timer_reg.read(timer_tmp , (bit<32>) meta.FlowID);
    timer_cnt=cur_time-timer_tmp;
    if(!IS_E2E_CLONE(standard_metadata)){
      if (cur_time>meta.pkt_thd ){
        timer_reg.write((bit<32>)meta.FlowID, timer_cnt);
        mark_packet();
        }
      if(meta.flag == 1){
          generate_clone.apply(); }
        }
    else{
              //fill telemetry header data
         }
  }
```

Fig. 1: Egress control flow of P4Flow.

matching based on the source and destination IP addresses provided by the control plane and assigns an ID to each flow, i.e., `FlowID`. Then, it checks if the packet belongs to that flow or it is a telemetry packet. In both cases, P4Flow forwards the packets according to the forwarding rules either to the destination of the flow or the controller. We assign a unique ID for each flow belonging to the traffic of different flows, while for all the telemetry packets we use one ID to save memory space of the switches.

Collecting telemetry. We use P4 registers to keep track of the flow statistics at the egress control flow. Each P4 register can keep the state of the flows in the memory of the switch. We get the timestamp of each packet from `egress_global_timestamp` metadata. Fig. 1 shows the part of the egress control flow implementation of P4Flow.

In Fig. 1, we define two registers, namely, `pkt_thd_reg` and `timer_reg`, to store the time interval threshold value to report the flow statistics and the timestamp of the packets for each flow, respectively. We initialize the `timer_reg` to zero to keep track of the flows using their `FlowID`. Upon receiving a packet of a flow, P4Flow updates the corresponding register value of `timer_reg` associated with that flow. The algorithm checks this value with a user-defined threshold value to send the flow information. To collect the statistics regarding the number of packets for each flow, we can use P4 `Counters` or `packet_length` metadata filed. P4Flow sends the flow information to the controller when the timer reaches the threshold value. We now explain the procedure of sending the flow telemetry.

Packet generation. There is no built-in mechanism in P4 to generate the packets, but we can use a copy of the current packet to send it to the desired egress port. One example solution is to clone the packet. Therefore, we use the clone `egress to egress` or `clone_e2e` primitive to mark the flow information on the packet header similar to [13]. The *mark_packet* action in Fig. 1 concatenates all the telemetry data to one bitstring and appends it to the cloned packet. This data field includes source and destination IP addresses, source and destination ports, and protocol fields to distinguish different flows. Additionally, we concatenate the last timestamp the telemetry updated and the size of transmitted data to this field. We define a one-bit metadata variable called `flag` to specify if the packet should be cloned at the egress control flow by applying `generate_clone` table.

IV. Evaluation

In this section, we first report the performance of our system on two custom graphs. Then, we measure the impact of the telemetry packets on the Flow Completion Time (FCT) on a real-network topology taken from [14].

A. *Performance on Custom Graphs*

We generate two graphs, namely, Waxman and Erdös-Rényi, each one with 200 nodes to compare the performance of different algorithms. We use $\alpha = 0.60$ and $\beta = 0.06$ as the values for the model parameters in generating Waxman graph in Python. Similarly, we set $p = 0.05$ as the probability of edge creation in Erdös-Rényi graph. Both graphs are widely used by the network community for research purposes. The Waxman graph has 644 edges, while the Erdös-Rényi graph has 1004 edges. We set each link latency value of the graphs by randomly selecting a value from the U.S network latency reported by AT&T [15]. Finally, we randomly assign a bandwidth value in Mbps for each link in the range of 10 to 40 Mbps. We generate 100 to 1000 uniformly random flows in which each flow f has maximum hops of six and assume that they are assigned using the traffic engineering techniques such as the ones in [2]. Therefore, P4Flow leverages the spare network capacity to steer the telemetry packets. We measure the reporting delays and the message overhead to the controller/collector. We also report the running times of the algorithms.

We compare the performance of P4Flow with those of OpenTM [7] and FlowCover [8] as two state-of-the-art flow monitoring tools. The network operators can adopt these approaches to monitor custom flows. However, several other works concentrate on the different aspects of flow monitoring such as coordinating the INT packets or monitoring all links.

OpenTM [7] considers the traffic matrix of the network and aims to minimize the number of telemetry packets. It offers different mechanisms for switch selection such as random, the last switch, and the least-loaded one. We implement the last switch selection in this work. FlowCover [8] aims to minimize the flow monitoring cost with high accuracy. It selects the switch that carries the highest number of flows in the network. To implement FlowCover, we check all switches in all the flows and pick the most frequent ones as the leaders. To have a fair comparison with OpenTM and FlowCover, we use the same P4 code of P4Flow to run the experiments but with the chosen leader nodes by each approach. Herein, selecting different values for the time interval can negatively impact the obtained results.

Reporting delay. Fig. 2 reports the flow statistics message delays for all systems on Waxman and Erdös-Rényi graphs. We take the sum of the link delay of each hop from the leader switches to the controller. The reporting delay of P4Flow on average is 1.2x and 1.4x less than FlowCover and OpenTM on Waxman graph (see Fig. 2a). This improvement on Erdös-Rényi graph is 1.57x and 1.71x compared with the other approaches since the graph has more edges (see Fig. 2b). The main reason for such an improvement of P4Flow is due to the parameters, e.g., inter-node delay, that considers in selecting leader switches.

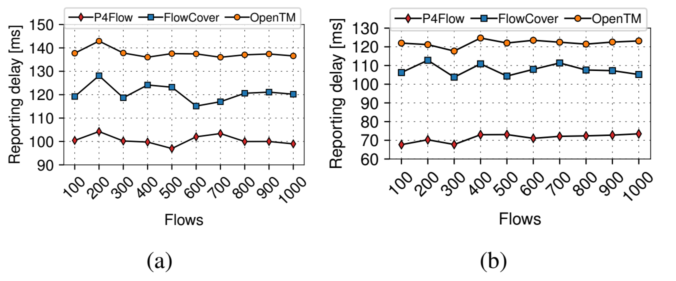


Fig. 2: The reporting delays of the systems on; a) Waxman graph. b) Erdös-Rényi graph.

Overhead. We compute the overhead of the telemetry packets for each flow $C(f)$ as follows.

$$C(f) = \mathcal{H}_f \times \mathcal{S}, \tag{2}$$

where $\mathcal{H}_f$ and $\mathcal{S}$ are the number of hops from the $\mathcal{L}_f$ to the controller and probe packet size, respectively.

Fig. 3 reports the flow statistics overhead of all systems for the same graphs. We use the same message size in all approaches to have a fair comparison. Fig. 3a shows that the message overhead of P4Flow on average is 1.63x and 1.78x less than FlowCover and OpenTM on the Waxman graph. We have similar improvement on Erdös-Rényi graph and P4Flow has 1.60x and 1.76x less overhead compared with the other approaches (see Fig. 3b). The main reason for such an improvement is that P4Flow uses fewer hops to send the flow statistics messages.

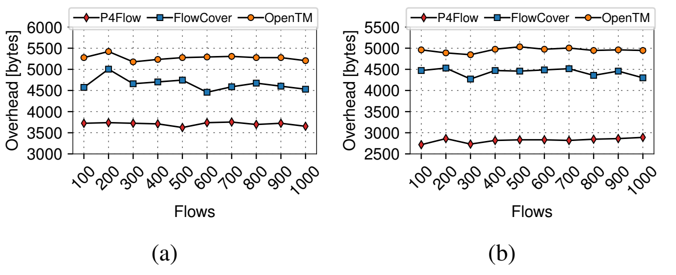


Fig. 3: The message overhead of the systems on; a) Waxman graph. b) Erdös-Rényi graph.

Running time. We now measure the running time of the leader selection algorithm in P4Flow, OpenTM, and FlowCover. Fig. 4 shows the running time of all systems for the same graphs. OpenTM selects the last node of a flow to report the flow statistics. Thus, its running time is constant and close to zero. The running time of FlowCover increases by increasing the number of flows because it has to check which node is the most repeated in all flows. P4Flow first ranks the node and then applies the leader selection algorithm to choose a node based on objective function in eq. 1. Therefore, its running time is higher than the other two approaches. However, this running time has less impact on the performance of the system since the network administrator can proactively execute it.

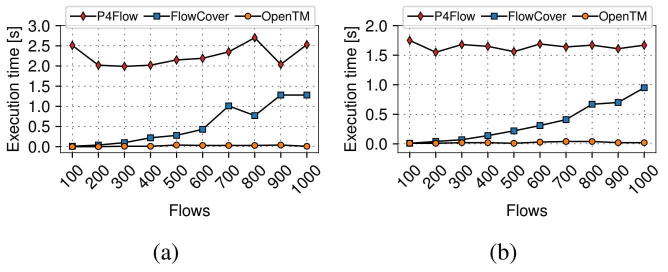


Fig. 4: The running time of the systems on; a) Waxman graph. b) Erdös-Rényi graph.

B. Performance on WAN Networks

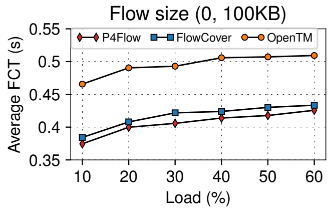
We now report the impact of flow telemetry on the FCT of different systems using an example wide-area network.

Testbed. We conduct the simulation using Mininet network emulator on a VM with an Intel Xeon CPU AMD Opteron(TM) Processor 6272 2.1GH with 32 GB RAM and 16 CPU cores running Ubuntu server 18.04. Mininet is the defacto emulation tool used by the P4 consortium to test the functionality of the P4 programs. We also apply the recommendation [16] of the P4 consortium to gain high throughput from BMv2 switches in Mininet.

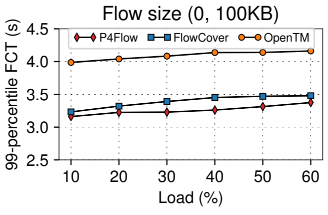
Workloads. We use one of the most popular empirically derived realistic workloads, i.e., FaceBook [17]. The traffic distribution is in this workload is heavy-tailed. We leverage the traffic generator in [18] to generate the desired flows according to a Poisson distribution of the workload and network load. The traffic generator generates traffic from 50 different sources that are randomly chosen from the network topology. The network load varies in the range 10% and 60%. The traffic generator can generate different flow sizes, but we limit them to the small size flows, i.e., flow size < 100 KB due to the limitations of our links in Mininet. However, this can be tuned according to the capacity of each link. We generate 1000 flows from each source with 0.8 ms as the threshold interval and report the averaged results over 10 different runs.

Topology. We compare the FCT of P4Flow with those of OpenTM and FlowCover on RocketFuel [14] topologies such as AS3967 as an example and report some representative results. We use the same links delay and bandwidth for these topologies since their exact values are unavailable.

FCT comparison. We now measure the impact of the leader nodes' selection on the FCT of flows. The goal is to understand the impact of the flows telemetry' overhead on FCTs. Fig. 5



(a)



(b)

Fig. 5: Comparisons of FCT for P4Flow, OpenTM, and FlowCover. a) Average FCT of small flows. b) the 99^{th} percentile of small flows.

shows average FCT and the 99^{th} percentile of flows of P4Flow, FlowCover, and OpenTM on AS3967. P4Flow improves up to 1.04x and 1.24x the FCT of flows over FlowCover and OpenTM, respectively. Furthermore, this improvement in the 99^{th} percentile FCT of flows is up to 1.08x and 1.29x compared with FlowCover and OpenTM. The results confirm that selecting proper leader nodes impacts the FCT of the flows in the network.

V. Practical Application Scenarios

In this section, we briefly explain some practical application scenarios that P4Flow can help. The first application scenario is geo-distributed streaming applications running on distributed processing systems such as Apache Flink. Many organizations leverage these systems to get invaluable insights from their customers by collecting information regarding the users' clicks on the websites or finding trends on social networks [19]. Depending on the queries running on this massive amount of data generated by the users, the network administrators can tune the time interval of collecting flow statistics. The other scenarios are video streaming or online gaming. The network operators can set the threshold value to gather the flow statistics depending on the sensitivity of the traffic flows. For example, delaying the flows in the video streaming scenarios can degrade the quality of experience (QoE) [20].

VI. Conclusion

In this letter, we proposed a network flow monitoring tool for programmable networks. Our tool can reduce the overhead of reporting the flow statistics by considering the properties of the inter-node links such as delay and bandwidth. The P4-based implementation of our approach showed that it can efficiently report the flow telemetry while improving the flow completion time. Furthermore, it allows the provider to tune collecting the flow statistics according to the applications' needs.

Acknowledgment

This work was partially funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A).

References

[1] Y. Zhou, C. Sun, H. H. Liu, R. Miao, S. Bai, B. Li, Z. Zheng, L. Zhu, Z. Shen, Y. Xi, P. Zhang, D. Cai, M. Zhang, and M. Xu, "Flow event telemetry on programmable data plane," in *SIGCOMM*, 2020, p. 76–89.

[2] H. Mostafaei, M. Shojafar, and M. Conti, "TEL: Low-latency failover traffic engineering in data plane," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2021.

[3] 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.*, vol. 44, no. 3, p. 87–95, Jul. 2014.

[4] The P4.org Applications Working Group, "In-band network telemetry (INT) dataplane specification v2.1," https://github.com/p4lang/p4-applications/tree/master/docs, 2020.

[5] J. Kučera, D. A. Popescu, H. Wang, A. Moore, J. Kořenek, and G. Antichi, "Enabling event-triggered data plane monitoring," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '20, 2020, p. 14–26.

[6] 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*, 2019, pp. 487–495.

[7] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic matrix estimator for openflow networks," in *Passive and Active Measurement*, A. Krishnamurthy and B. Plattner, Eds. Springer Berlin Heidelberg, 2010, pp. 201–210.

[8] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "FlowCover: Low-cost flow monitoring scheme in software defined networks," in *2014 IEEE Global Communications Conference*, 2014, pp. 1956–1961.

[9] A. G. Castro, A. F. Lorenzon, F. D. Rossi, R. I. T. d. C. Filho, F. M. V. Ramos, C. E. Rothenberg, and M. C. Luizelli, "Near-optimal probing planning for in-band network telemetry," *IEEE Communications Letters*, vol. 25, no. 5, pp. 1630–1634, 2021.

[10] R. Hohemberger, A. G. Castro, F. G. Vogt, R. B. Mansilha, A. F. Lorenzon, F. D. Rossi, and M. C. Luizelli, "Orchestrating in-band data plane telemetry with machine learning," *IEEE Communications Letters*, vol. 23, no. 12, pp. 2247–2251, 2019.

[11] E. Triantaphyllou, *Multi-criteria decision making methods*. Springer, 2000.

[12] J. Young and T. Barth, "Web performance analytics show even 100-millisecond delays can impact customer engagement and online revenue," 2017, Akamai Online Retail Performance Report, https://bit.ly/3xznKyT.

[13] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: Practical real-time microburst monitoring for datacenter networks," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, ser. APSys '18, 2018.

[14] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, p. 133–145, Aug. 2002.

[15] ATT Global IP Network Latency, "U.S. network latency," http://soc.att.com/3sfHMxr.

[16] P4 Language Consortium, "Performance of bmv2," 2020, https://github.com/p4lang/behavioral-model/blob/main/docs/performance.md.

[17] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15, 2015, p. 123–137.

[18] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling {ECN} in multi-service multi-queue data centers," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 537–549.

[19] H. Mostafaei, S. Afridi, and J. H. Abawajy, "SNR: Network-aware geo-distributed stream analytics," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2021, pp. 820–827.

[20] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017, p. 197–210.