One-Way Delay Measurement From Traditional Networks to SDN: A Survey

DJALEL CHEFROUR, Mathematics and Computer Science Laboratory, University of Souk-Ahras, Algeria

We expose the state of the art in the topic of one-way delay measurement in traditional and in software defined networks. A representative range of standard mechanisms and recent research works, including OpenFlow and P4 based schemes, are covered. We classify them, discuss their advantages and drawbacks; and compare them according to their application environment, accuracy, cost and robustness. The discussion extends to the reuse of traditional schemes in SDN and the benefits and limitations of latter with respect to reducing the overhead of network wide measurements. We conclude with a summary of learned lessons and open challenges for future work.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Networks** → **Network measurement**; *Time synchronization protocols.*

Additional Key Words and Phrases: one-way delay measurement, active and passive techniques, software defined networks, time synchronization

ACM Reference Format:
Djalel Chefrour. 2021. One-Way Delay Measurement From Traditional Networks to SDN: A Survey. *ACM Comput. Surv.* 54, 7, Article 156 (July 2021), 35 pages. https://doi.org/10.1145/3466167

1 INTRODUCTION

This survey is motivated by the relevance of the *one way delay (OWD)* and the recent advances in the techniques that measure it, especially in datacenters and software defined networks (SDN). Measurement is a fundamental building block of network management which is in general lagging behind other areas of networking research [71]. In particular, there is a big need for network delay measurement as it is of major importance to:

- Network equipment vendors who must comply to strict specifications about the router forwarding delay, for instance, to implement Fast Channel Change for IPTV or ensure a high Quality of Experience (QoE) for VOIP in residential and business gateways.
- Network managers who exploit the delay information for various tasks, such as: traffic engineering, load balancing, routing, anomaly detection (identifying high latency causes [96]), security analysis (*e.g.,* detecting Denial of Service (DoS) attacks [57]), Quality of Service (QoS) provisioning and Service Level Agreement (SLA) validation.
- Delay sensitive applications which experience failure or degradation of their performance when the delay is high. Usage scenarios include: interactive applications such as e-learning and videoconferencing [16]; and real-time applications in the fields of: the internet of things (IoT) [29], high-performance computing and high frequency trading [45].

Author's address: Djalel Chefrour, djalel.chefrour@univ-soukahras.dz, Mathematics and Computer Science Laboratory, University of Souk-Ahras, P.O. Box 1553, Souk-Ahras, Algeria, 41000.

0360-0300/2021/7-ART156 $15.00
https://doi.org/10.1145/3466167

- An important family of transport protocols that depend on the delay as an early signal of congestion to optimize their throughput (*e.g.,* Swift [47] and TCP-LP [2]); or to back-off and avoid delaying other flows (*e.g.,* LEDBAT [76]).
- A new type of Active Queue Management (AQM) disciplines that aim at solving the bufferbloat problem [31]). For instance, Controlled Delay (CoDel) [65] limits the time spent by packets in the routers buffers.
- Analytics servers that are fed with delay information coming from streaming telemetry. They use it, for instance, to train neural networks [28] for the prediction of the average the delay and for routing optimization.

The topic of OWD measurement gained attraction over time thanks to the wide expansion of the Internet, the advent of faster networks and the spread of time sensitive applications. However, in some cases the OWD was estimated by measuring the round trip time (RTT) and halving it. Such a trivial approach does not give an accurate result when the forward delay is different from the backward one, which is the case when there is asymmetry in: routing [67], queuing [3] and link speeds (*e.g.,* Asymmetric Digital Subscriber Loop (ADSL)). Therefore, many research works concentrated on the measurement of the OWD using other approaches.

Another motivation of this survey is the lack of a recent literature review dedicated to OWD measurement, especially in SDN. As we will show in the related works (Section 3), the existing surveys about network metrics measurement put the emphasize on traffic statistics. That is collecting packet and byte counters at the flow and port levels. However, time-dependent metrics like the delay are more challenging to measure. Moreover, the per-flow delay is harder to estimate compared to the aggregate delay since the number of flows can be quite large. For instance, up to 100 K flows for a switch forwarding 1.2 Tbps of traffic [40].

SDN offers new ways to measure the delay and further possibilities to exploit this information, thanks to their logically centralized control and their programmability. The measurement of many interacting flows in traditional IP networks is usually run by the end hosts independently. This might cause a high overhead on the network while the benefit (of knowing the delay) is limited to these end hosts only. By contrast, estimating such flows delays in a centrally controlled and scalable way should lower the network overhead and generalize the information benefit.

Beyond the problem of time synchronization between the vantage points, the estimation of the OWD raises the general issue of trade-off between the measurement overhead and its accuracy. In particular, the forwarding nodes usually have limited high speed memory and processing power to carry on the measurement at high line rates. For instance, they do not necessarily implement hardware packet timestamping, nor support natively multiplication and division instructions [8] required for the calculation of the average OWD. Additionally, the communication channel between these nodes and the control and management ones can become a performance bottleneck in the case of high streaming telemetry. The latency of this channel might also influence the results accuracy if it is involved in the measurement process. Furthermore, there is the issue of granularity, that is the ability to zoom-in on the delay at the packet and flow level; and express such requests easily in network monitoring queries.

We classify OWD measurement into active and passive schemes; and analyze and compare them – where relevant – according to the following usage criteria: applicability, accuracy, cost and robustness. The applicability determines in what type of network a technique can be used. Particularly, it specifies what portions of the network are assessed and what is exactly measured. We denote this information as the *delay structure*. The accuracy is expressed in terms of the relative error in measurement ($|true\ value - estimated\ value|/true\ value$). The cost includes any special hardware required by a technique along with its storage, processing and network overhead. The

cost determines also the scalability of a technique. The robustness concerns the correct handling of the network effects (asymmetry, packet loss, reordering and duplication), which could bias the results.

The contributions of this paper include:

- An detailed analysis of OWD measurement techniques in traditional IP networks and their comparison with respect to the previous usage criteria (Sections 4.1 and 4.2).
- A study of SDN architectural elements relevant for OWD measurement (Section 5.1).
- A thorough review of OWD measurement schemes in SDN (Sections 5.2 and 5.3) and also their comparison according to usage criteria (Section 5.4).
- A discussion and comparison of OWD measurement schemes between traditional networks and SDN (Section 5.5).
- A summary of learned lessons and a list of original insights into future research works (Section 6).

Before detailing our subject, we clarify the terminology used throughout this article in Section 2, then we review the related works and define the scope of this survey in Section 3.

2 TERMINOLOGY

The IETF adopted a standard definition of the *one-way delay (OWD)* in RFC 2679 [41]. We draw on this definition which we summarize as: the duration required for a packet to completely travel between two hosts in a network, measured at wire-time from the transmission of the first bit to the reception of the last one. Considering wire-time over host-time has the advantage of excluding any extra latency induced by the hosts, especially if the packets timestamps are captured in software.

The terms *delay* and *latency* are often used interchangeably in the literature. Throughout this survey, we use the term *delay* in short to refer to *the one-way delay* unless explicitly specified. We reserve the term *latency* to point out any undesirable extra delay introduced by a network component even if it is unavoidable.

The OWD can be decomposed into four parts: *processing*, *queuing*, *transmission* and *propagation* delays [12]. The first two components are variable whereas the second two are deterministic, *i.e.,* they remain constant for a fixed path and packet size. Many measurement techniques rely on this decomposition. They try to isolate the deterministic delay from the variable part either to measure the delay itself or estimate the clock skew between network nodes.

The processing time is usually negligible compared to the three other components, therefore it can be ignored. The transmission latency is determined by how fast the bits are serialized on the communication medium. It depends on the traversed links speeds and is also negligible for rates above 10 Mbps. The delay of propagation, which happens at the speed of the electromagnetic wave in the given environment, can be ignored too, except for long distance links. The queuing latency is the most complex component of the delay as it varies with the traffic load and the capabilities of the forwarding nodes.

In terms of network coverage, the delay can be measured between: end hosts, any two nodes or adjacent nodes. In the latter case, we talk about *link delay* or *one hop delay* interchangeably. Link delay includes the processing and queuing latencies of the source node, in addition to the transmission and propagation delays. It is useful to know when debugging the network performance, for instance to pinpoint the origin of delay spikes. Conversely, the applications are more interested in the *end-to-end one-way delay* [48]. In contrast to the previous IETF network-centeric definition of the OWD, the end-to-end delay includes in addition the source host processing time, so what is measured is closer to what is experienced by the end user. Nonetheless, the end-to-end OWD is also used by the network operators and vendors to refer to the first definition above.

The end-to-end OWD is very often considered at the flow level. A *flow* is typically defined by the five-tuple formed by: the protocol, the ports and the IP addresses of the source and the destination [27]. Multiple flows belonging to different applications can share the same path or segment in a network (*i.e.,* a traffic mix). Conversely, one flow can take different paths due to load balancing techniques like equal cost multipath routing (ECMP). In this context, it is important to distinguish per-flow end-to-end delay from close but different metrics such as *flow duration* (which is also referred to as *flow completion time*) and *path delay*.

On the one hand, a flow duration is the period between the beginning of transmission of its first packet and the end of the reception of its last packet. This metric can be measured approximately in traditional networks by subtracting the flow start time from its finish time. For instance, such times are captured by NetFlow [18] in the records of the first and the last switches of the path. Flow duration is proportional to the end-to-end delay but can not be used to estimate it directly. Nonetheless, the flow start time at different switches along the path is used for this purpose by some of the works reviewed here.

On the other hand, the *path delay* between end hosts can be used to express the mean time needed for packets belonging to a certain flow to reach one node from another. But it can also refer to the OWD at the aggregate level, that is the average delay of a mixture of flows co-existing on the same path. The distinction is in the granularity. In general, different flows on the same path can have distinct delay properties, since they can traverse separate priority queues. Moreover, flows belonging to the same QoS class or even the same application type can have individual average delays different from their mean aggregate delay. This is due to their burstiness [50] or to the heterogeneity of their statistical behavior (*e.g.,* video with different activity levels) [80].

Finally, we do not consider per-packet delay separately from the per-flow delay. Estimating a flow's mean delay resorts to measuring the delays of all of its packets or at least a sample of them. Some applications like DNS query and real-time bidding depend on the delay of one lonely packet [49]. Additionally, if the packets of one flow are load balanced across multiple paths, the delay of each sub-flow can be considered independently. In general there is no temporal delay correlation across paths [73, 74]. Nevertheless, load balancing using multipath routing is rarely performed at the packet level. Instead, it is done on a per-flow or per-destination basis [5].

3 RELATED WORK AND SCOPE

There is an exhaustive literature about the topic of one-way delay measurement, including some review papers. In the following, we investigate the works that are close to ours and emphasize the difference with them. Then we draw the limits of our survey in the scope subsection.

3.1 OWD measurement in traditional networks

The tutorial from Svoboda *et al.* focuses on the definition of packet delay and its influencing factors [82]. They propose a well defined methodology to measure the delay using active, passive or hybrid approaches. In the case of active measurement, the attention is drawn to the careful selection of the probes sizes and their distribution in time. Both parameters must be randomized to avoid any possible bias. This tutorial serves as good introduction to the issue of delay measurement with few illustrations from wireless cellular networks. Therefore it is not meant to review the related research works thoroughly.

The survey by Bajpai and Schönwälder addresses delay estimation as one among many topics covered by internet performance measurement platforms and standards [7]. Hence, they considered few delay measurement examples in a brief way (OWAMP and RIPE TTM). Unlike ours, their goal was to illustrate and not to cover a broad multiplicity of works.

The surveys from Devito *et al.* [23] and Orgerie *et al.* [66] concentrated on the usage of GPS, PTP, NTP and TSCclock [90] to measure the delay. However both surveys did not consider the techniques that do not require clock synchronization. The latter approximate the delay after the detection and removal of clock skew from traffic traces. They are covered as part of the broad time synchronization topic by Wang *et al.* [91], Shin *et al.* [77] and Chefrour [15].

Many OWD measurement techniques bypass the issue of time synchronization altogether. They use the clock of one network node only (*e.g.,* RTT based and cyclic paths schemes) or consider the delay within one forwarding node (*e.g.,* RLI). To the best of our knowledge, there is no comprehensive survey that reviewed such techniques, despite their numbers and the time elapsed since their publication. Our survey fills this gap by covering a representative range of mechanisms from all the classes mentioned above.

3.2 OWD measurement in SDN

Some literature surveys reviewed the network measurement mechanisms in SDN. They classified them into different categories and highlighted their typical usage in network applications (*e.g.,* traffic load balancing and QoS provisioning). For instance, the review by Yassine *et al.* proposed an interesting categorization of several SDN traffic measurement techniques [92]. This categorization is based on the balance made between accuracy, on one side; and measurements overhead and usability for real-time decision making, on the other side. Only one work among all the reviewed ones concerns OWD measurement (*i.e.,* OpenNetMon). However, it was not discussed with respect to this metric. The review highlight was rather put on how OpenNetMon overhead is limited by decreasing probing when the flow rates stabilize.

Akyildiz *et al.* addressed the topic of network monitoring as part of traffic engineering in SDN [1]. They pointed out that SDN benefited from the flow statistics collection mechanisms developed in the traditional IP networks (*e.g.,* NetFlow and sFlow). But it suffered from their high overhead on the controller in the case of large datacenter networks. This limit lead to the development of new techniques that use adaptive polling or push based estimations. Nonetheless, the measurement of time-dependent statistics was considered in this review as an open research challenge, though some SDN works about the delay existed already.

The survey by Shu *et al.* organized SDN network measurement in three directions: network parameters measurement, a generic measurement framework; and traffic analysis and prediction [78]. The authors discussed examples of research works for all types of parameters with respect to used mechanisms, overhead and precision. Once more, regarding the delay, only OpenNetMon was reviewed. Similarly, Karakus and Arjan dedicated a section to network monitoring mechanisms in their survey about Quality of Service in SDN [43]. Again, as far as the delay is concerned, only OpenNetMon was discussed. It was deemed disadvantageous for the controller because of its messages overhead.

The review from Tsai *et al.* [87] proposed a decomposition of network monitoring close to the one from Shu *et al.* [78]. The authors compared SDN monitoring steps (*i.e.,* measurement and processing) with traditional IP networks; and showed that adaptability and flexibility of SDN can improve monitoring tasks. The collected data usually includes byte and port counters used for traffic engineering tasks. Although the reviewed works did not include the delay, flow management in real time was mentioned as an open issue for future research.

To summarize, these surveys provided the overall picture of SDN traffic measurement. They mostly covered volume statistics collection through OpenFlow (*i.e.,* byte and packet counters at port and flow levels). However, they did not provide a thorough review of the metrics that are more difficult to estimate such as the delay. In perspective, our survey complements them by putting

emphasize on one important metric, which is the one-way delay. We have not found any survey dedicated to OWD measurement in SDN so far.

3.3 Scope of this survey

Our primary focus is one-way delay measurement in traditional and in software-defined networks. We do not address this topic in the context of wireless networks because of the important number of research works there. This is due in part to the grand variety of wireless technologies (*e.g.,* Bluetooth, WLAN, wireless sensors and cellular networks). Covering all of them requires a separate survey.

Many mechanisms among the ones we reviewed rely on network time synchronization between the measurement points. However, we do not detail this topic here due to the extensive literature that concerns it. We dedicated a separate survey for network clock synchronization; and the detection and removal of clock skew from traffic traces [15].

Precising how applications and network operators would benefit from knowing the delay is beyond the scope of this study. Moreover, we do not consider in this paper how the end-to-end delay can be improved by using SDN compared to traditional networks. Some surveys about SDN include examples of how it lowers the end-to-end delay. For instance, the survey from Hafeez *et al.* [34] addresses the advantages of SDN enabled congestion control, which has a direct impact on the queuing delay.

We do not cover other network performance metrics such as throughput, round trip time (RTT) and jitter (delay variance). First, a flow throughput is easy to infer once its delay and its volume are known. Second, RTT is easier to measure than the delay as it can be estimated by one node only. Nevertheless, it does not highlight the difference between the forward and backward paths. Some applications, like bulk data transfers and multimedia streaming, depend on the OWD more than RTT. Last, jitter calculation is straightforward if the delay (at least the inter-packet one) is tracked over a period of time. However, some applications are susceptible to jitter changes (*e.g.,* VoIP and multimedia streaming), but others are not (*e.g.,* HTTP).

The interest in network delay does not stop in its measurement. It extends to modeling it for the purpose of prediction, which is useful in QoS provisioning. We do not address delay modeling in this paper as it mandates its own survey. Nevertheless, we will see in some of the reviewed works that statistical models of the delay (*e.g.,* Poisson and Gamma distributions) are used to infer its variable part and to pace active probing.

4 OWD MEASUREMENT IN TRADITIONAL NETWORKS

The OWD measurement techniques in traditional IP networks have been classified in two categories: active and passive based on whether they use test probes or not [82, 91, 97]. Hereafter, we review a representative range of techniques from each category. We compare them using a rich table format and discuss their relative advantages and limitations. Next, we consider how can we reuse them in SDN.

4.1 Active OWD measurement

Active measurement relies on sending timestamped probes from a sender to a receiver. The stream of probes is typically generated in a random way to avoid bias, following a Poisson [94], a Gamma [6, 14] or a Geometric process [81]. Bias can happen if successive probes are separated in time by a fixed interval and coincide with a periodic network activity (*i.e.,* a phase lock). If the sender and the receiver clocks are synchronized, each probe OWD is obtained by subtracting its reception time from its transmission timestamp. Otherwise, some scheme is used to eliminate the clock offset between the measurement points.

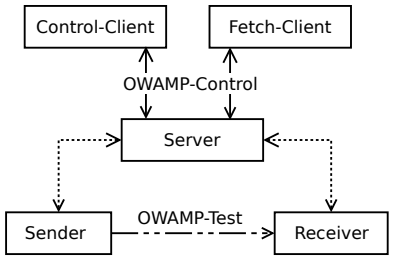

Fig. 1. Logical components of the One-Way Active Measurement Protocol (OWAMP) [94].

4.1.1 One-Way Active Measurement Protocol. The IETF standard methodology of OWD measurement is defined in RFC 4656 about the One-Way Active Measurement Protocol (OWAMP) [94]. It builds on the OWD definition from RFC 2679 [41] and uses Poisson sampled probes. Figure 1 illustrates the logical components of OWAMP, which include in addition to the Sender and the Receiver of the probes: a Control-client that requests the measurements, a Server that schedules, starts and stops the test sessions; and a Fetch-client that collects the results. The latter components communicate using the OWAMP-Control protocol layered over TCP. The dotted arrows in Figure 1 indicate an implementation dependent communication between the Server and the measurement points. The dashed arrow indicates the OWAMP-Test protocol that defines the probes format over UDP. It contains a timestamp and a sequence number, which is used to detect packet loss, reordering and duplication.

The tools `owping`/`owampd` [39] provide a client server implementation of OWAMP. `owping` implements the Control-client, the Fetch-Client and the Sender, while `owampd` hosts the Server and the Receiver. Pathak *et al.* used these tools in the Internet to study delay asymmetries and reported delays from tens to hundreds of milliseconds [67]. Other active measurement tools very close to OWAMP, though prior to it, include Surveyor [42] and RIPE TTM [24]. The latter used Endace DAG cards to capture the probes wire-times and obtain the ground truth of the OWD.

4.1.2 SLA Monitors. The SLA Monitor (SLAM) from [81] is an active probing tool that uses NTP Stratum-0 synchronization and kernel level timestamping for OWD measurement. It follows a methodology that differs from OWAMP in how the mean delay is calculated. SLAM models the delay as a continuous function $f(t)$, where f is the probe OWD and t its transmission time. It divides the measurement period into adjacent sub-intervals containing each three probes: two endpoints a, b and a midpoint c. The lengths of these sub-intervals ranging from 5 ms to 500ms are drawn from a geometric distribution. The delay of a sub-interval j is calculated using Simpson' numerical integration of the delays of its three probes (*i.e.,* $\frac{1}{6}(f(a_j) + f(b_j) + 4f(c_j))$). The delays of all sub-intervals are weighed with their lengths and summed to obtain the total area under the delay function. This total is divided by the number of the sub-intervals to yield the mean OWD. SLAM gives a delay estimation more accurate than OWAMP when both are tested in the same setup (with similar probe rates) and compared to the ground truth. The latter is obtained by passive monitoring using DAG cards.

The Cisco IP SLAs feature also measures the OWD in both directions between a Source and a Responder [19]. Both nodes add transmission and reception timestamps to the probes they exchange and need to be clock synchronized with NTP. The Responder listens to control messages from IOS SLA operations that specify the transport protocol (UDP or TCP) and port of the probes. These control messages can use MD5 authentication for added security. The OWD measured by IP SLAs

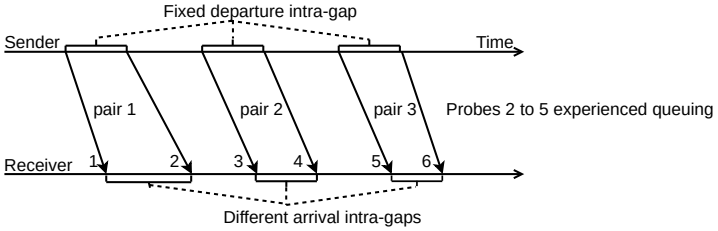

Fig. 2. Packet pair probing to assess queuing delay.

can be visualized and monitored for threshold violation via popular tools like Network Performance Monitor and PRTG Network Monitor.

4.1.3 RTT based measurement. The OWD can be actively estimated without synchronizing the sender and the receiver clocks. The most straightforward method to do so is to measure the RTT based on the sender clock and halve it as done in NTP. However, this method is not accurate if the forward and reverse paths are delay asymmetric. Choi and Yoo propose a scheme that overcomes this limitation by measuring the RTT at both ends with a continuous message exchange [17]. The sender transmits a new probe as soon as the ACK of the previous one is received and measures the RTT from its side. The receiver measures also the RTT using its own clock by comparing the transmission times of the adjacent ACKs. The forward and backward OWDs are then derived analytically from the sender and receiver RTTs as the clock offset they contain is canceled in the equations. A separate estimation of the initial forward delay is performed heuristically from the first two exchanges to bootstrap the derivation. Such a step has a big impact on the accuracy of the following estimations. Moreover, this method works only if the clock offset between the end points is constant.

4.1.4 Packet pair technique. Another way of achieving active delay estimation without clock synchronization is the packet pair technique from Lu *et al.* [57]. It transmits periodically a pair of packets separated by a fixed time interval called *intra-gap*. This technique is based on the observation that, for a fixed packet size and path, probes should experience the same deterministic part of the delay. Hence, an arrival intra-gap that differs from the departure one indicates a discrepancy in the variable part of the OWD, which is the queuing delay (see Figure 2). If the departure intra-gap is large enough so that the two probes do not co-exist in the same queue at each hop, then their queuing delays can be considered as independent random variables. Therefore, the probability density function (PDF) of the queuing delay can be approximated. This is realized by performing an iterative fast Fourier-to-time reconstruction algorithm on the measured intra-gaps. The mean queuing delay over a period of measurement is then calculated as the expectation of its PDF. However, this technique does not measure the deterministic part of the OWD.

For the evaluation of the packet pair method, the authors used an intra-gap of 5 ms and an inter-gap (*i.e.,* departure interval between the probe pairs) of 14 ms. In a simulated 10 hops LAN, their method converges to the actual queuing delay (*i.e.,* 86 ms) after 10 iterations of the PDF reconstruction algorithm. In a real campus LAN, the PDF reconstructions took 1.67 s on average for queuing delays below 20 ms.

4.1.5 Cyclic-path method. Gurewitz *et al.* propose a cyclic-path (CP) method for the estimation of all the links OWDs in a network without clock synchronization [33]. CP measures first the link OWD between each pair of neighbors in both directions using timestamped probes. It assumes that

the network is not permanently overloaded and the clock offset is constant. Thus, the minimum observed OWD for each link is considered as the deterministic delay inflated or deflated by the neighbor's relative clock offset. Second, the clock skew is removed from these measurements using literature techniques. Lastly, the clock offsets are removed by solving a constrained optimization problem. The constraints are derived from the observation that, along a cyclic-path, summing the minimum measured links OWDs eliminates the clock offsets of their nodes and gives the deterministic OWD of this path. That is the sum of the unknown variables representing these links delays is equal to the known measured path delay. The set of constraints is constructed by finding as many independent cyclic-paths as possible. In a connected network with V nodes and E links, there are at maximum $E - (V - 1)$ independent cycles. They can be formed by finding a spanning tree and building cycles using links inside and outside this tree. The complexity of this operation is $O(Elog(V))$, but it happens only once if the topology does not change.

The authors propose two objective functions for the optimization problem. The first is the least square error (LSE) method that consists of minimizing the error between the real links delays and their estimates. The second is the maximum entropy (ME) method, which is based on a probabilistic model of the link deterministic delay. It estimates the probabilities of finding a hypothetical probe packet that is hopping randomly in the network on each directional link. This probability is proportional to the link deterministic delay. The longer the delay of a link, the higher the probability to find the packet on that link. Given the lack of knowledge about the real delays, the least biased estimation of the corresponding probabilities is provided (according to information theory) by the maximum entropy probability distribution [33].

Regarding the variable part of the delay, the authors consider that it has the same distribution of the initial OWD measurements but shifted with a constant. This constant is composed of the deterministic part of the delay and the constant clock offset. In the simulation based evaluation of the cyclic-path method, the authors assumed that the delay follows an Erlang distribution. They used 20 nodes connected with a mixture of symmetric and asymmetric 102 links and eight NTP like probes per link. The deterministic OWDs of the asymmetric links are estimated by both objective functions (LSE and ME) in a way much better than the simple halving of RTT. ME gives slightly superior results than LSE with a maximum absolute error of 2 ms for random delays. Nevertheless, the cyclic-path method cannot estimate the OWD between any two nodes separately. It is unsuitable for online measurements because it does not yield an instantaneous result.

4.1.6 Constrained cyclic-path method. Vakili and Gregoire propose an improvement of the previous method to reduce its time complexity and increase its accuracy, especially if the network nodes are far apart (*e.g.,* transatlantic links) [88]. In addition, this improvement can also measure the OWD between two nodes without mapping the delays of all the links in a network. This is achieved by using more constraints for the original cyclic-path LSE optimization problem. These constrains are obtained by measuring four types of delays: (*i*) the RTT between the two measurement points, (*ii*) the RTT between each one of them and a third node, (*iii*) the RTT of a cyclic-path formed by all three nodes and (*iv*) the transmission delay between the measurement points in both directions by using the packet pair technique (with an intra-gap of 0.1 s). The measurement of the RTTs is carried in small time intervals (*i.e.,* around 200 ms) where the clock skew is negligible, whereas the measurement of the transmission delay assumes a constant clock skew of 1 part per million (ppm) in average.

On the one hand, limiting the cyclic-path to a third-party node reduces the time complexity. On the other hand, the constraints based on the transmission delay increase the accuracy of the solution further than using constraints derived from the propagation and the processing delays. This is due to the fact that the former delay is asymmetric while the latter two are not. Moreover,

to avoid the effects of queuing delay on the result of the packet pair technique, two interleaved pairs with different probe sizes are sent instead of one pair. Compared to the previous technique, the better accuracy of this one comes at the cost of an increased number of probes and a longer convergence. The transmission delay of tens of milliseconds magnitude is estimated correctly after tens of seconds of calculation. Hence, this technique too is not suitable for online measurements.

4.1.7 Reference Latency Interpolation. Lee *et al.* propose a method called Reference Latency Interpolation (RLI), for the per-flow forwarding delay measurement inside and between adjacent routers [50]. It is based on the observation that packets from different flows exhibit an important temporal similarity within short queue bursts (below one second). Therefore, the router measures only a reference stream of packets that are closely spaced in time with the target flow. Then it interpolates the measurements linearly to obtain the flow's forwarding delay. The reference packets are timestamped probes injected between the router ingress and egress ports. They would experience the same treatment as the regular packets. The delay obtained inside the router is composed of the processing and queuing latencies and does not need time synchronization. Additionally, RLI can be implemented between adjacent routers synchronized with PTP to obtain the link delay, which includes the transmission and the propagation latencies.

RLI probes rate is adapted dynamically according to the link utilization. The estimation of the latter requires a byte counter per port, whereas the interpolation of the delay requires three hardware counters per flow. The reference probes' overhead is below 0.1% at low link utilization and decreases further when the latter grows. However, RLI does not give the end-to-end OWD as individual router delays are not easily summable. This is because none of the sender, the receiver and the individual routers knows the path of the flow. The latter might even change if multipath routing is used. Moreover, deploying RLI at all the routers along a path has an important cost.

4.1.8 Comparison and Discussion. We compare the OWD active measurement techniques above with respect to: delay structure, probing process and accuracy; and cost in Tables 1, 2 and 3, respectively.

The test probes used by active techniques constitute an additional load on the network. So their size and pace must be chosen carefully to avoid a negative impact on other network traffic [6]. Particularly, for long distance links, the transmission delay is relatively smaller than the propagation one and contributes little to the overall delay. Hence, the packet size can be small if active probing is used [81].

Concerning the accuracy of a technique, although we use its relative error as an indicator, it should be considered only in the context of a given delay structure and magnitude. The latter depends on the type and the topology of the network where the measurement occurs; and on the traffic patterns that materialize there. Some of the reviewed works confirmed that the relative error varies when these parameters change (*e.g.,* SLAM and CP).

The cost of a technique includes also any special hardware that it requires for time synchronization and stamping (*e.g.,* GPS receivers or router modifications); and its time complexity. If the complexity is not constant, the technique will require a certain time to converge and therefore needs to be considered carefully for online use.

Owping, Surveyor, RIPE TTM and the packet pair technique include all a sequence number in their probes. While this has a negligible cost, it allows at the same time the assessment of packet loss, reordering and duplication. In addition, it makes the delay measurement itself robust to such network effects.

The OWD obtained with active probing is meaningful for one type of traffic only, *i.e.,* the probes type. It cannot be generalized to different classes of application traffic. Furthermore, active probing

Table 1. The delay structure of OWD active measurement techniques in traditional networks

Technique	Test Network (speed)	Coverage	Components
Owping	Internet (PlanetLab n/r)	end-to-end	total
Surveyor	Internet (T3 54 Mbps)	end-to-end	total
RIPE TTM	Internet (n/r)	end-to-end	total
SLAM	Fast LAN (OC3 155 Mbps)	end-to-end	total
RTT based	NS2 (ADSL 1280/128 Kbps)	end-to-end	total
Packet pair	NS2 & Campus LAN (100 Mbps)	end-to-end	queuing
Cyclic-path	Simulated WAN (n/r)	all links	deterministic
Constrained CP	NS2 (ADSL 464/123 Kbps)	per link	total & transmission
RLI	YAF simulator (10 Gbps*)	per hop	processing + queuing

Legend: *=with a CAIDA real traffic trace from an OC-192 10 Gbps Tier-1 ISP link

Table 2. The probing and accuracy of OWD active measurement techniques in traditional networks

	Probing			Accuracy	
Technique	**Process**	**Rate (pps)**	**Size (bytes)**	**Delay (ms)**	**Error**
Owping	Poisson	10 per path	≥ 14	> 10	n/a
Surveyor	Poisson	2 per path	40	> 60	n/a
RIPE TTM	Exponential	0.025 per path	128	> 150	0.2% *
SLAM	Geometric	0.048 per path	100	~ 40	0.3% *
RTT based	Fixed	2 per RTT	40	250	0.5% *
Packet pair	Fixed	2×50 per path	60	86	2% *
Cyclic-path	Fixed	8 per link	48	10–40	n/r
Constrained CP	Fixed	40 per link	107	100	1%
RLI	Adaptive	0.1% of link usage	n/r	0.1–30	$< 11\%$

Legend: pps=packet per second, n/a=not available, n/r=not reported,
*=approximated from absolute error

Table 3. The cost of OWD active measurement techniques in traditional networks

Technique	Synchronization	Timestamping	Time Complexity
Owping	NTP	kernel	$O(1)$
Surveyor	GPS	kernel	$O(1)$
RIPE TTM	GPS	user space	$O(1)$
SLAM	NTP Stratum 0	kernel	$O(1)$
RTT based	not needed	user space	$O(1)$
Packet pair	not needed	not needed	$O(Nlog(N))$
Cyclic-path	not needed	user space	$O(N)$
Constrained CP	not needed	user space	$O(N)$
RLI	not needed	hardware	$O(B)$

Legend: N=# probes, B=interpolation buffer size, $B < N$

is able to detect delay spikes but cannot pinpoint the flow(s) causing it. Comparatively, passive measurement does not suffer from these two disadvantages.

Another limitation of active measurement is the risk of dropping or delaying the probes beyond network administrative boundaries. This might happen if there is no prior agreement between the operators. They might flag each other's probes as malicious traffic or might want to hinder the competition. The administrative constraints need to be taken into account when selecting the transport protocol and the destination port number of the probes.

Lastly, some active OWD measurement techniques assume that the minimum observed delay is the deterministic one (*e.g.,* CP). Therefore, they are not usable when there is long-lasting queuing which throws off this assumption . Likewise, the techniques that assume that the delays of the probes are independent random variables (*e.g.,* packet pair) do not work properly where this assumption does not hold. That is if the minimum time gap between consecutive packets is not larger than the worst congestion event. In such a case, these two packets might co-exit in the same congested queue and the first packet delay will influence the second one.

4.2 Passive OWD measurement

Passive measurement consists of capturing the properties of traffic existing in the network. Zseby *et al.* investigated the main elements required for this task, which are timestamping, efficient packet filtering and unique identification (ID) generation [98]. The goal is to match the packets that traverse the measurement nodes, so their delay can be calculated from their transmission and reception times. For the sake of simplicity, the measurement nodes are called sender and receiver even if they are not the original source and destination of the traffic. The delay calculation can be performed at the receiver or at a third node. So the measurement data (*e.g.*, IDs and timestamps) needs to be carried there by control messages either in-band (using the same network under test) or out-of-band (using a different network) [98].

The best accuracy is obtained with synchronized hardware monitors that capture wire-times at the measurement points. However, this is also the most expensive solution. Corvil devices which are used in trading markets are an example of such monitors [21]. In general, when the sender and receiver clocks are not synchronized, skew estimation and removal schemes are applied to the traffic trace. Otherwise, GPS is used for clock synchronization in WAN, as in the early work of Graham *et al.* [32]; and Precision Time Protocol (PTP) [25] is used in fast LAN, as in many of the works detailed below.

4.2.1 Lossy Difference Aggregator. Kompella *et al.* suggest the router primitive Lossy Difference Aggregator (LDA) for the measurement of delays in the order of microseconds within datacenters [45]. LDA measures the delay of a path inside a router where the input and output ports are considered respectively as the sender and the receiver that share the same clock. It can also measure the delay of a link between adjacent routers synchronized with PTP. The sender and the receiver maintain each an array of pairs of *timestamp accumulator–packet counter*. They both sample traffic using the same hash function to map a packet to an accumulator-counter pair. The sampling rate is proportional to the size of this array and inversely proportional to the number of packets and their loss rate during a measurement interval of one second.

The use of multiple accumulator-counter pairs is equivalent to splitting the traffic into sub-streams. Hence, it limits the effect of packet loss on the counters to a subset of packets instead of the whole aggregate. The sender and receiver LDAs are compared after every measurement interval of one second. This yields the packet loss from the mismatching counters and gives the average delay from the matching ones by subtraction of their accumulators and division over these counters.

The simulation based evaluation of LDA shows that the error in delay estimation increases with the loss rate. Compared to active probing in similar settings, LDA gives more accurate estimation of the delay with little network overhead. Active measurement would require a high probing frequency to detect the same fine-grain OWD. Nonetheless, LDA is not resilient to packet reordering as it requires FIFO processing to sample the same set of packets at both measurement points. Additionally, it cannot measure per-flow delay nor the end-to-end delay.

4.2.2 Consistent NetFlow. The Consistent NetFlow (CNF) architecture from Lee *et al.* calculates the delay of a flow's first (or last) packet between two NetFlow routers by subtracting its timestamp in the first NetFlow router from its timestamp in the second one [51]. It assumes that the routers are synchronized with PTP or GPS and that there is no packet reordering. CNF modifies NetFlow routers to add hash based sampling according to RFC 5475 [62]. Its aim is to alleviate the measurement burden on high speed networks and to ensure that consecutive routers on a given path sample the same flows. When a flow expires, its records are exported by NetFlow towards a central node. The latter collects the records of many routers and matches the ones belonging to the same flow. It also performs timing checks to eliminate inconsistencies due to packet loss and the difference in processing and queuing latencies between the routers.

Each consistent sampled flow will have four timestamps: two in the sender for the first and the last packet; and two similar ones in the receiver. CNF calculates the OWD of a small flow (< 4 packets) as the average of the first and last packet delays. This method is called the *Endpoint* estimator. For a large flow, the calculation builds on the observation that the delays of packets that experience the same queuing are correlated. Thus, CNF estimates a large flow OWD as the average of the delays of all the packets which have a sender timestamp between the sender timestamps of this flow's first and last packets. This method is named the *Multiflow* estimator. Its relative error decreases with the flow size and is unbiased by packet loss rates below 5%.

4.2.3 FineComb. Lee *et al.* address the measurement of fine-grain aggregate delay in datacenters with a mechanism named FineComb [52]. It divides time into intervals of few seconds delimited by *sync* control messages sent from the sender to the receiver. For each interval, both nodes agree on a subset of packets for which they measure the delay using the same hash function. Each node maintains an array of M buckets similar to LDA's accumulator–counter pairs. However, FineComb buckets include in addition an incremental packet stream digest to cope with the problem of packet reordering. Near the interval ends, reordering can through out the measurements because a packet that starts in one interval at the sender might drift to another one at the receiver. If there is a mismatch between the sender and the receiver digests, the corresponding bucket is qualified as useless. Nonetheless, it can be recovered by stashing some information at the receiver. The stash memory is made of the timestamp, the bucket index and the incremental digest of a small number of packets that surround each *sync* message. It is used to attempt the recovery of a useless bucket by subtracting one of the stashed digests from this bucket's digest.

The authors define the optimal stash size W as the one that maximizes the expected number of good samples. The latter are sampled packets that do not suffer neither loss nor reordering which cannot be recovered using the stash. So $W = \rho N p$ where ρ is the packet reordering rate, N is the number N of packets transmitted by the sender during a measurement interval and p is the sampling rate. However, the authors choose to fix M and W based on the memory available at the receiver; and select the corresponding optimum sampling rate p. For this purpose they approximate the packet loss as a Poisson random variable which has the rate β; and therefore formulate p as:

$$p = \min\left\{\frac{M}{2\rho^2 N}\left(2\rho + \beta - \sqrt{4\rho\beta + \beta^2}\right), 1\right\} \tag{1}$$

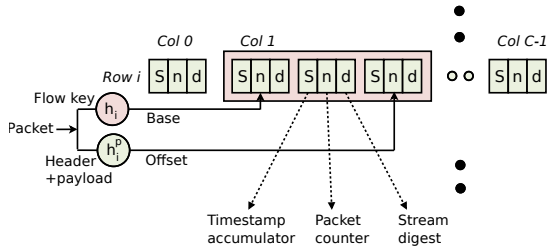

Fig. 3. Lossy Difference Sketch (LDS) data structure [73].

Simulation shows that FineComb performs better than LDA even under high packet loss and reordering rates. It has a mean relative error of the delay estimate around 0.1%. Experiments on the stash recovery time using a 2.33 GHz processor show that it takes no more than 14 ms, even for a high stash to bucket ratio (W/M).

4.2.4 Lossy Difference Sketch. Sanjuàs-Cuxart *et al.* present the Lossy Difference Sketch (LDS) mechanism that measures per-flow latency at microsecond granularity [73]. It combines LDA with sketches to measure a flow delay without maintaining per-flow state. The sketch in question is a matrix of accumulator–counter–digest buckets. Every sampled packet is mapped in parallel at all the rows of this matrix. At each row, mapping happens first to a base column using a hash of the flow key, then to a random cell in the neighborhood of the base column using a hash of the full packet (see Figure 3). Each matrix cell gives (by dividing the accumulator over the counter) an estimate of the weighted average delay of the flows that mapped to it, where the weights correspond to the count of packets of each flow. The spreading of a flow's packets across many cells limits the impact of packet loss and reordering on the flow counters, whereas the mapping to multiple rows reduces the impact of flows collision in the same cell.

LDS estimates a flow's OWD from the subset of its useful matrix cells which have the minimum counters to avoid the interference of large flows. It uses an LDA like sampling scheme but with an increasing rate per row to protect against high packet loss. The authors evaluated LDS using a real traffic trace captured by DAG cards synchronized with PTP. Compared to RLI and Multiflow CNF in the same test settings, LDS gives a more accurate delay estimation with a lower memory usage and is robust to loss rates below 1%. It is also robust to packet reordering as its second hash function parses the full packet. However, FineComb is more robust to packet reordering than LDS due to its stash recovery scheme.

4.2.5 Measurement Architecture for Packet LatEncies. The Measurement Architecture for Packet LatEncies (MAPLE) from Lee *et al.* works (like LDA) inside and across high speed routers synchronized with PTP [49]. MAPLE uses a hardware data structure based on Bloom filters [11] to approximate the latencies of all the packets in a scalable and fast way. Its idea is to cluster packets based on their delay and store only the center of each cluster using a pair of timestamp accumulator–packet counter. This reduces the high speed memory need to 12.8 bits/packet. This memory is dumped to secondary storage at the end of 1 s long measurement intervals. The clusters centers can be selected statically in linear time using a logarithmic scale (*e.g.,* 0.1-1μs, 1-10μs, etc.). Alternatively, they can be selected dynamically using the K-means algorithm, which gives a better accuracy because it minimizes the distances between the members of each cluster and its center. Nevertheless, its complexity is super linear. Hence, the authors use hybrid clustering by selecting half of the clusters statically and the other half dynamically.

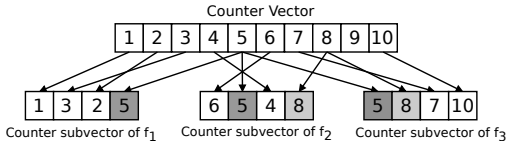

Fig. 4. Counter based Per-flow Latency Estimation (COLATE) [75].

MAPLE contains also a query engine that allows clients to query routers about the OWD at the packet and sub-flow level. This makes it the finest technique in terms of delay granularity. For that purpose, the query uses a hash of the packet or a concatenation of the flow's IP identification fields to reduce the query bandwidth. The latter is inversely proportional to the flow size in terms of number of packets. As a result, queries of packet ranges are compressed to an average ratio of 6% (*i.e.,* 17 times less bandwidth usage) compared to queries of individual packets. For the evaluation of MAPLE, the authors used 50 delay centers ($K = 50$) as a good balance between accuracy and complexity. However, this parameter remains platform dependent.

4.2.6 NetFlow profiling. Kögel proposes a method to measure per-flow OWD from the data collected and exported by standard NetFlow routers at the edge of an enterprise network [44]. Like CNF, this methods uses the NetFlow timestamps of the first (and the last) packet of a flow. It also assumes that routing is symmetric without packet loss. However, it performs no sampling and does not require clock synchronization between the measurement points. Instead, the latter are profiled using traffic captured during a light network load. The goal of this profiling is to detect the routers relative clock offset and skew. So we call this technique NetFlow Profiling (NFP). The clock offset and skew are used later, when the OWD is calculated, to adjust the timestamps obtained from the NetFlow records. The experimental evaluation of NFP showed the existence of a standard deviation around 30 ms in the estimation of delays ranging from 50 ms to 200 ms. This error is due mainly to the limited timestamp resolution of the tested routers (*e.g.,* 64 ms for Cisco 6500 series and 4 ms for Cisco 7200 series). Hence, the usage of this technique is restricted to wide area networks.

4.2.7 Counter based Per-flow Latency Estimation. Shahzad and Liu propose a Counter based Per-flow Latency Estimation scheme (COLATE), which uses PTP but does not require the timestamping of packets [75]. COLATE records packet times at each measurement point in a vector of n counters. Contrary to the previous works, we note that the term counter here refers to a timestamp accumulator, not a packet counter. Each flow is mapped with a collision free hash function to a random subset of counters called a *subvector* of size m (see Figure 4). This avoids the overflow problem for elephant flows if a single counter is used per-flow. A counter might be shared by multiple subvectors to save memory. Therefore it might contain a mix of timing information from other flows, which is considered as noise. Upon transit, each packet is mapped to a counter in its flow subvector and the current time is added there. Before any counter overflows, the vector is dumped from SRAM to permanent storage, then reset to zero. This yields a *counter epoch* delimited by the timestamps of the first and last recorded packets. The maximum value of the timestamp accumulator per epoch is denoted T.

To estimate the OWD, COLATE finds all the counter epochs that overlap with the flow's starting and ending times. Then, it uses statistics schemes to de-noise the timing information and estimate the total of the timestamps contributed only by the flow's subvector. For this purpose, the authors use either the expectation based estimator (EBE) at runtime or the maximum likelihood estimator (MLE) offline. The former has a higher variance but is computationally simple (*i.e.,* $O(m)$). The latter has a small variance but involves more complex computations (*i.e.,* $O(m2^b)$ where b is the

Table 4. The delay structure of OWD passive measurement techniques in traditional networks

Technique	Test Network (speed)	Coverage	Granularity
LDA	Simulated datacenter link (10 Gbps*)	per link & in router	aggregate
CNF	Backbone (10 Gbps)	between NetFlow routers	per-flow
FineComb	Simulated datacenter link (10 Gbps**)	between routers	aggregate
LDS	ISP link (10 Gbps)	per link & inside router	per-flow
MAPLE	Simulated datacenter link (10 Gbps**)	per link & inside router	per-packet
NFP	Global enterprise network (n/r)	between NetFlow routers	per-flow
COLATE	Simulated datacenter link (10 Gbps**)	between routers	per-flow

Legend: *=synthetic traffic trace, **=same real traffic trace used by RLI (cf. Table 1)

Table 5. The accuracy and robustness of OWD passive measurement techniques in traditional networks

		Accuracy		Robustness	
Technique	**Sampling rate**	**Delay(μs)**	**Error**	**Loss**	**Reordering**
LDA	$M/2(lN+1)$	0.2	0.2%	✓	
CNF	0.1% to 1%	20–100	20% *	✓	
FineComb	see 1	10	0.2%	✓	✓
LDS	like LDA, differ. l/row	50	1.2% *	✓	✓
MAPLE	1%	< 100	n/r	✓	
NFP	100% (no sampling)	50–200 ms	n/r		
COLATE	100% (no sampling)	< 100	5% **		

Legend: M=# accumulator–counter pairs, l=packet loss rate
N=# packets per interval, n/r=not reported, *=median, **= at 95%ile

size of each counter in bits). Lastly, comparing the timestamp accumulators of a given flow at two measurement points and dividing by the flow's number of packets yields its OWD.

While COLATE assumes that there is no packet loss, its accuracy can be tuned to the required degree by configuring the parameters: b, n, m and T. The Matlab based simulation of COLATE with EBE shows that it has a relative error of 5% at 95% confidence level, where $b = 12$, $m = 20$, $n = 455334$ and $T = 8 \times 10^{10}$. Its SRAM consumption amounts roughly to 1 MB per measurement point.

4.2.8 Comparison and Discussion. Tables 4, 5, 6 and 7 summarize and compare the previous OWD passive measurement schemes, respectively, according to: delay structure, accuracy and robustness, hardware data structures and cost. The delay measured by all these schemes is the total one, so we omit the delay components column in the first table. As for the active mechanisms, the relative error of the passive techniques should be considered in the context defined by the network type and the patterns of the measured traffic. But also with respect to other parameters, like the sampling rate and network effects (*i.e.,* packet loss, reordering). The latter parameters have also an impact on the robustness of these techniques (*e.g.,* LDA and FineComb).

In general, most passive techniques present the advantage of not interfering with regular traffic, but are tributary of its presence in the network. They reflect the delay of real data packets rather than artificial probes which might be treated differently in the network. However these schemes raise privacy and security issues as they need to parse the packets contents. They also need to be deployed at selected points in the network which is not always possible.

Table 6. The hardware data structures of OWD passive measurement techniques in traditional networks

Technique	Hardware Data structure
LDA	M pairs of timestamp accumulator–packet counter
CNF	pN NetFlow records with start and end timestamps
FineComb	M buckets of accumulator–counter–digest + W stash entries
LDS	R rows matrix with cells of accumulator–counter–digest
MAPLE	Shared Vector Bloom filters of K clusters and counters
NFP	N NetFlow records with start and end timestamps
COLATE	Vector of timestamp accumulators of b bits each

Legend: p=sampling rate, N=# packets or flows per interval, where $W < M \ll N$, $K \ll N$ and $R \ll N$

Table 7. The cost of OWD passive measurement techniques in traditional networks

Technique	Control rate	Synchronization	Complexity
LDA	72 Kbps	PTP	$O(M)$
Endpoint CNF	n/r (NFX)	PTP or GPS	$O(1)$
Multiflow CNF	~	~	$O(pN)$
FineComb	n/r	GPS	$O(M2^{W/M})$
LDS	n/r	PTP	$O(R)$
MAPLE	1/17*	PTP	$O(Klog(pN))$
NFP	n/r (NFX)	clock skew removal	$O(N)$
EBE COLATE	n/r	PTP	$O(m)$
MLE COLATE	~	~	$O(m2^b)$

Legend: see Table 6 for M, p, N, W, R and K, NFX=NetFlow export rate, m=# accumulators per flow, *: see subsection 4.2.5 for details subvector

For high traffic rates, the storage and processing of the measurement data (timestamps, counters, etc.) constitutes an important overhead. All the reviewed works manage this overhead by modifying the routers hardware, which adds an extra cost. For instance, some mechanisms use hardware aggregate data structures (*e.g.,* LDA) to reduce the SRAM bill. However, this affects the granularity of the measurement which is then limited to the aggregate level. Some methods use hardware traffic sampling (*e.g.,* CNF) to avoid dealing with all the packets. Nevertheless, this lower the accuracy of measurements, pose the risk of missing the packets of short lived flows and inflate the effect of packet loss and reordering if not handled carefully.

Depending on the granularity of each technique, traffic sampling can be performed at the packet level (*e.g.,* LDA and MAPLE) or at the flow level (*e.g.,* CNF and COLATE). Few of the reviewed techniques reported the hash function they used for sampling (*e.g.,* SHA-1 for FineComb and H3 or BOB for MAPLE). So we do not consecrate a column for this information in the cost table. Nonetheless, we refer to Henke *et al.*'s empirical evaluation of numerous hashing schemes that can be used for OWD measurement [37].

Regarding NetFlow based techniques, it is worth noting that the router processing of a flow's first packet can take longer than the the following ones. The first packet has to go through the so called *slow path*, where it travels high in the network protocols stack to get a routing decision. The latency

of this costly step varies according to the router software scheduling. The routing decision is cached by the lower layers to be reused for the fast forwarding of the flow's next packets, called the *fast path*. This caching happens usually in hardware, *i.e.,* in the application-specific integrated circuit (ASIC), to keep up with line rates. Therefore, the first packets timestamps exported by the routers will have a delay inflated by the slow path latency and will not reflect the flow's average delay. This can be tempered by considering the last packet timestamps or the times of all the packets that are temporally correlated with the target flow as done in Endpoint and Multiflow CNF, respectively.

Concerning robustness, we note that among the reviewed works none addressed the issue of packet duplication, so we omit its column from the corresponding table. Duplicate packets can bias the result of the mechanisms that use aggregate data structures (*e.g.,* LDA, FineComb, LDS, MAPLE and COLATE), because their transit times might be counted repeatedly (as many times as there are duplicates) in the timestamp accumulator. However, as these techniques use traffic sampling, it is very unlikely that two or more copies of the same packet are sampled together, because they are closely spaced in time.

5 OWD MEASUREMENT IN SDN

Before detailing OWD measurement in SDN, we present here a brief review of the SDN architecture. Next, as with traditional networks, we categorize OWD measurement in SDN into active and passive techniques. We cover a representative selection in each category, compare them together and discuss their advantages and shortcomings. Then, we analyze the similarities and differences between OWD measurement in traditional networks and in SDN.

5.1 SDN architecture

The SDN paradigm is based on the separation of the network control plane from its data plane [46]. The former is removed from the forwarding nodes (*i.e.,* switches and routers) and put in a logically centralized controller. Similarly, the applications that use and manage the network are separated in their own plane. Figure 5 shows a typical SDN architecture where the control plane is based on OpenDayLight [58] and ONOS [9], which are two widely used service oriented SDN frameworks. The depicted data plane follows the protocol independent switch architecture (PISA) popularized by the domain specific language for Programming Protocol-independent Packet Processors (P4) [13]. One the one hand, the applications exchange with the control plane via different northbound APIs, such as the Representational state transfer (REST) ones. On the other hand, the communication between the two lower planes uses southbound APIs like the OpenFlow protocol [83], NETCONF [26] and the gRPC Network Management Interface (gNMI) [85]. These APIs make the configuration of the data plane malleable, especially the forwarding tables, and allow the efficient and scalable monitoring of its operational state. For the network operators and the developers, SDN has two main advantages. It provides a central global view of the network and makes it more flexible [27].

In traditional networks, the information about the topology, the links state and the nodes statistics is scattered among the forwarding nodes, while in SDN it is gathered available at one logical place. Hence, decision making based on this information (*e.g.,* routing, management, monitoring and measurement) becomes easier as it is decoupled from the information collection process. However, if this collection is performed via frequent polling it might constitute a burden on the managed devices and on the collector itself in large networks. So it is better achieved in real-time with push based streaming telemetry mechanisms, such as gNMI and NETCONF, which are scalable, extensible and vendor-independent. Furthermore, modifying the forwarding nodes behavior, for instance to add a packet field for the sake of measurement, requires an upgrade of all the intermediate nodes, which usually faces resistance from the vendors and the administrators. By contrast, a similar change is achievable in SDN via the modification of the controller or the P4 program alone, which

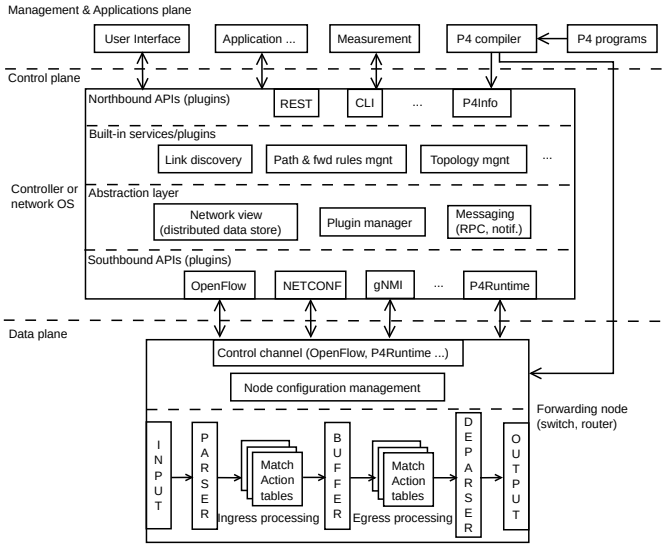

Fig. 5. SDN architecture and its influence on packet delay.

is less tedious and less costly. Thus, traditional networks are ossified, whereas software defined ones are programmable.

Figure 5 shows also the SDN elements that have an influence on packet delay and are therefore relevant for its measurement. When a data packet arrives at a programmable switch, its header is parsed to extract fields which are sent to the match-action tables. The entries of these tables contain: matching rules based on the header fields, actions to perform (*e.g.,* drop, forward, duplicate, modify a header etc.) and collected statistics (*e.g.,* per-port and per-flow byte and packet counters, flow duration, etc.). Matched packets take the fast path and their forwarding happens at the line rate. Otherwise, the switch asks the controller what to do using a control protocol such as OpenFlow. The awaiting packet is then handled through the slow path and this increased its delay. The parser, the match-action rules and the control flow between the tables are all programmable using a high level language such as P4. The latter includes in its metadata the packets ingress timestamps which is captured by the parser and is useful for the measurement of time-dependent metrics such as the delay.

In the case of OpenFlow, the forwarding tables are populated by the controller with *FlowMod* messages that installs flow entries based on the routing decision. Unmatched packets are notified to the controller using a *PacketIn* message. The flow entries that are not used expire after a certain idle timeout and are notified to the controller via a *FlowRemoved* message. The controller can also query the switch counters using OpenFlow statistics messages and inject packets in the data plane using the *PacketOut* message. Since its inception, the OpenFlow specification defined the *EchoRequest* message that carries a timestamp and can be emitted by the controller or the switches. This timestamp is to be returned in the *EchoReply* message, which allows the measurement of the control channel RTT. Version 1.5.0 of OpenFlow introduced the *scheduled bundles* feature to group large numbers of flow entries modifications and commit them in several switches at a certain execution time [83]. This feature requires the synchronization of the controller and switches' clocks using NTP or PTP. Scheduled bundles request and reply messages might carry a transmission timestamp, which is used by the controller to check if a switch's clock is synchronized with its own.

It is also used by a switch to roughly estimate its clock offset relative to the controller. However, this timestamp is not currently used for network delay measurement.

Another SDN southbound API that can benefit delay measurement is the IETF network configuration protocol NETCONF and its companion data modeling language YANG [10]. The former is an RPC based protocol that uses XML encoded messages to control network devices. The latter defines a generic (vendor-neutral), extensible, tree-based and strongly typed data model to describe the state and configuration of these devices. In particular, Lee *et al.* propose an extension of YANG that includes a `<one-way-delay>` metric for traffic engineered networks [53]. This metric can be queried with the NETCONF `<get/>` operation or monitored in the context of streaming telemetry using the YANG push extension [20], but the authors do not specify how it is measured. Likewise, Mizrahi *et al.* propose in RFC 7758 [61] and extension of NETCONF with a *Time Capability* that allows the scheduling of RPC requests and the reporting of their execution time in a forwarding node. This extension is similar to OpenFlow *scheduled bundles* and is also not used to measure OWD for the time being.

Device state polling and streaming telemetry in SDN can also be realized with gNMI. This API can encode its messages in JSON, protocol buffers and ASCII. It is defined by the OpenConfig group of network operators, which specifies also several YANG modules for programmable network devices. Particularly, the OpenConfig `probes` module contains tests that send actives probes at fixed intervals, to measure the RTT between a source and a destination using various protocols (ICMP, UDP, TCP or HTTP) [84]. Its test results (*e.g.,* `min/max/avg-delay` expressed in microseconds) can be obtained in a pull or push-based way with gNMI `Get` and `Subscribe` requests, respectively. The latter is used for instance by the `gnmi_collector`. Nonetheless, as of the time of this writing, the gNMI/OpenConfig framework does not support the OWD measurement. Though, this can be added easily as its data models include already the management of NTP servers, which allow the clock synchronization of the measurement points.

5.2 SDN active OWD measurement

A common feature of active delay measurement in SDN is the sending of test probes over a loop that includes the controller. The latter starts the probing and collects the final results. We discuss hereafter how this is handled in different research works.

5.2.1 Controller in the loop. Phemius and Bouet were among the first to use OpenFlow for the measurement of the link OWD between adjacent SDN switches [68]. The SDN controller sends a timestamped probe to the first switch using the *PacketOut* message. This probe is an Ethernet frame which source address is the egress port attached to the target link. Thus, the first switch forwards this frame to the second one, which sends it back to the controller using a *PacketIn* message, because the frame's destination address is the broadcast one. To obtain the link OWD, the controller captures the probe reception time and subtracts from it: its timestamp, its own processing overhead and the latency of the control channel to each switch. So we call this technique *Controller in the Loop (CL).* The controller processing overhead depends on its capabilities and therefore needs to be calibrated per platform by measuring the latency of an unused link. The latency between the controller and each switch is measured by halving the RTT of the *Statistics* request and reply messages they exchange, assuming the control channel is delay symmetric.

The CL technique was evaluated using Mininet [59] and Linux's `tc` tool to simulate link latencies from 0 to 20 ms. It has an average relative error of 1%, compared to halve the RTT measured by `ping` in the same setup. It does not require neither time synchronization, nor special forwarding rules in the switches tables, nor network hardware modifications.

The OWD of a path containing several switches is composed from the measured delays of all its links. These are added to the processing overhead which is multiplied by the hop count. However, as the per-link probes do not travel down the path consecutively, they do not experience the exact same delay as the normal data packets, especially if there is highly variable queuing. Indeed, the egress port over which a probe is sent to controller is different from the port over which data packets are forwarded to the next switch. Typically, each egress port has its separate output queue. Moreover, this technique cannot measure the end-to-end delay accurately as it does not include the latency between each end host and its close by switch.

5.2.2 Controller in the LLDP loop. Liao and Leung present an OWD measurement mechanism that is similar to the previous one, but uses LLDP packets instead of OpenFlow messages carrying special probes [54]. So we name it *Controller in the LLDP Loop (CLL)*. LLDP packets are usually injected in the data plane at fixed periods by the SDN controller to discover the network's topology. The authors modified their format to add a type-length-value (TLV) field, which encapsulates a timestamp of four bytes. When a switch receives these packets, it floods them to its neighbors which send them back to the controller. The latter captures the LLDP packet reception time and subtracts the control channel latency to measure the link delay. This mechanism incurs no network overhead as LLDP is used for topology discovery anyway. Hence, it can be viewed as a passive method. But we choose to present it with the active ones as the LLDP packets are not application data packets.

The CLL method shares most of the advantages and the drawbacks of the previous one. However, it avoids the overhead of special probes. For its evaluation, the authors used a Mininet linear topology and `ping` based RTT halving as a base line. They obtained relative errors of 20, 5 and 3%, for link latencies of 0, 1 and 1 ms, respectively. This result suggests that the measurement error depends on delay magnitude and that the controller in the loop techniques are not suitable for measuring sub-millisecond latencies.

5.2.3 Many data loops. Altukhov and Chemeritskiy propose an improvement of the controller in the loop technique to eliminate the bias of the control channel latency [4]. The latter can be bigger than the path delay by many orders of magnitude. We choose to call this scheme *Many Data Loops (MDL)*, because its idea is to loop the probe many times between the first and last switches of the target data path. At the start of looping and periodically, after a certain number of iterations, the first switch sends a pulse message containing the number of elapsed iterations to the controller. This is achieved by modifying the switches forwarding tables with special rules using OpenFlow *FlowMod* messages. The controller can then divide the time between the reception of successive pulse messages by the number of iterations to obtain a more accurate measurement of the path RTT. This scheme has the advantage of avoiding the latency of the control channel as the pulse message is sent once the probe is inside the data path. It also takes into account the queuing latency in the data path.

For the experimental evaluation of MDL, the authors used a path that traverses three virtual switches, which were running inside a real OpenFlow switch with 1 Gbps ports. The controller and two traffic generators were running in a separate server, which has three NICs linked with the real switch. The measurement of a reference average RTT of 540 μs was obtained by generating 1000 byte timestamped probes over the path of interest. The MDL estimation of this path's RTT gave a mean relative error of 3.7%, which means that this technique has an accurate estimation, especially that the delay is in the sub-millisecond scale. However, the overhead of looping the probes on the network is quite high. The estimation of the previous RTT required a loop of 1024 iterations.

The MDL technique can derive a link OWD from its measured RTT by halving it, then taking into account the queue length at the port of the source switch in each direction. The queue length

Q is approximated over a period of congestion T from the port byte counter X and the link capacity C. That is $Q = X/C - T$ (in units of time). Nevertheless, this works only with links where the transmission and propagation latencies are respectively equal in both directions. MDL can also calculate the path OWD by summing the OWDs of its links. However this has the drawback of exacerbating the network overhead further. Especially, when the OWDs of many paths need to be measured in parallel. Additionally, the controller would become a performance bottleneck because of the high number of pulse messages it will receive. In such a case, the overhead can be alleviated by avoiding redundant measurement of the links shared between many paths.

The authors propose to map the delays of all the links in a network by selecting a minimum number of independent long loops that cover these links. This is somehow similar to the cyclic-paths technique discussed earlier. To avoid the exponential explosion of a search based on finding the graph spanning trees, they use a greedy algorithm to maximize the length of elementary loops by combining them. Still, the number of forwarding rules per switch required by this method is quite high. It is proportional to the number of iterations per loop and the number of switches per path. For instance, around 100 rules in total are necessary to map the links delays of a five nodes graph with a low density.

5.2.4 OpenNetMon. Van Adrichem *et al.* present a monitoring extension to the POX OpenFlow controller called OpenNetMon [89]. It injects probes of 72 bytes at the first switch of a flow path and redirects them back to the controller when they reach the last switch. The controller calculates the OWD using the same approach of the first technique above. However, the test probes are injected over a separate VLAN, which connects the controller and the switches directly. This scheme avoids the uncertainties caused by the software scheduling of switches when they use *PacketOut/In* messages.

Special forwarding rules are used to forward the test probes which are paced at and adaptive rate. The latter is proportional to the path throughput and is obtained by polling the path edge switches (the first and the last one) to collect their packet counters and compare them. The rate of this polling is also adaptive. It increases when flows arrive or change their throughput and decreases when their statistics stabilize. So the probing overhead on the network grows with its load for a better delay accuracy. Similarly, the polling overhead on the control channel augments to converge quickly to an accurate throughput result when the network conditions change. The experimental evaluation of OpenNetMon used Open vSwitch (OVS) instances running on Intel Xeon servers with interconnections limited to 100 Mbps. It exhibited a mean relative error of 2.3% in the estimation of a OWD delay of 7 ms, which was pre-configured using the Linux `NetEm` tool [36].

5.2.5 TTL based looping. Sinha *et al.* suggest an improvement of the MDL technique above by limiting the number of forwarding rules to only three per switch [79]. They restrict the loops to one link and exploit the IP time to live (TTL) field to count their iterations. Therefore, we call this scheme *TTL based Looping (TL)*. The controller installs in every switch three rules that: (*i*) decrement the TTL of the probe at the source switch and forward it to the next one, (*ii*) loop it back to the first switch when it reaches the destination switch and (*iii*) send it up to the controller when TTL becomes zero. The latter divides the time difference between the emission of the probe and its reception by the number of iterations to calculate the link delay. A path OWD is measured by repeating the looping over all its links. Hence, this method suffers from the same drawback of the previous mechanisms that measure the delay per link. That is it does not mimic the exact behavior of the data packets traveling down the consecutive links. Its relative error was not reported.

5.2.6 TTL based LLDP looping. Liao *et al.* propose a method that combines the previous TL technique with their former CLL mechanism discussed above [55]. In this this method, which we

name *TTL based LLDP Looping (TLL)*, the LLDP packet is looped three times over a link between two switches. This looping is hard-coded to avoid the use of special forwarding rules. However, this requires a switch modification and is not a flexible solution. The timestamping of the packet happens in the link destination switch when it arrives there for the first time. This switch adds a 10 bytes long TLV for that purpose, then sends the packet back to the source switch. When the looping ends at the destination switch, it captures the packet final reception time, calculates the link RTT, stores it in the packet and sends it up to the controller. Hence, TLL avoids the measurement uncertainty caused by the control channel latency altogether. However, as with all RTT halving techniques, the OWD cannot be inferred accurately this way on asymmetric links or when there is different queuing in both directions. Additionally, TLL has a network overhead much higher then the original TL technique because each LLDP packet is by default flooded to all the neighbors of a switch.

With regard to network wide delay measurements, the authors formulate a Vertex Cover Problem (VCP) over the un-directed graph of a network to measure all its links. That is they select the switches that will loop the LLDP packets by finding a minimum vertex cover. This has the benefit of minimizing the overhead on the control and data planes. For tree-based topologies, which are common in datacenters, the authors solve the VCP with a greedy algorithm that is better than the exponential exhaustive search.

Emulation with Mininet shows that the error in RTT estimation is inversely proportional to its magnitude. The estimation of RTTs below 1 ms exhibits an error of 10%, which confirms that the measurement techniques driven by the controller are in general not suitable for such time scales.

5.2.7 Software-defined Latency Monitoring. The framework for Software-defined LAtency Monitoring (SLAM[1]) from Yu *et al.* aims at measuring the OWD between any two switches in a datacenter continually [93]. SLAM's controller pre-configures the data plane with rules that forward the test probes along the switches of the path to measure. It injects one probe per second in the first switch using *PacketOut* messages. The probes trigger at the last switch *PacketIn* messages, which are timed by the controller to estimate the path OWD distribution over a period of time. SLAM refines the captured delays by removing both the control channel latency and the processing times of each edge switch. It measures the former latency constantly using *Echo* messages, while it monitors the latter with a *PacketOut* message that generates a *PacketIn* response without forwarding any probe.

For the evaluation of SLAM, the authors added a separate physical connections between the edge switches and the controller, as if the latter was part of the data plane. Their goal is to measure a reference path OWD distribution in addition to the one estimated via the control channel. This is achieved by programming the edge switches to mirror the probes over these separate connections. The controller calculates the reference OWD from the reception times of the mirrored probes. So in total it ends up with two OWD distributions which it compares using the Kolmogorov-Smirnova statistical test. This test, which is linear in time, shows a match between these two delays for the millisecond scale and above. However, SLAM overestimates the OWD for sub-millisecond latencies because of the time uncertainties introduced by the control channel. Especially, due to the varying processing times at the switches and the controller.

5.2.8 SDProber. Ramanathan *et al.* propose the SDProber mechanism for OWD measurement in SDN [69]. It generates test probes along the path between the measurement points. These points mirror the probes towards a collector node which compares their reception times to estimate the OWD. SDProber varies its probing rate per link between lower and upper bounds defined by the network operator. When measuring all the links of a network, it constrains the total number of

[1]This is different from the work with the same acronym in Section 4.1.2. We keep the one chosen by the authors.

probes to limit their overhead. Then it computes the shortest paths covering all the links while satisfying these constraints. As this computation is an NP-hard, the authors propose to use a pseudo random walk on the weighted directed graph of the network. The links weights are used in the switches forwarding rules when selecting the next link to be visited by a probe. A high weight augments the probability of visiting the associated link in the next iteration of the random walk.

SDProber uses binary exponential backoff after each iteration to adapt the weights and keep the probing rates between the configured bounds. Specifically, it halves the weights of the links which have normal delays while it doubles the weights of un-inspected links. Likewise, it halves (doubles) a weight when it crosses the low (high) link bound. In addition, the weight of a link that suffered high latency recently is multiplied by a higher factor to achieve a faster detection of congestion. The weights adaptation is carried indefinitely as long as monitoring continues. This raises a question about the convergence of the random walk (*i.e.,* covering all the links) when monitoring periods are short. The *cover time* of the random walk, that is the number of steps required to visit every node, is polynomial $O(n^3)$ in terms of n the number of nodes [56].

The authors evaluated SDProber using a Mininet topology made of 196 nodes and 243 links, an iteration period of 30 seconds and different probe rate bounds (up to min-max rates of [20-16] per minute). They showed that the random walk keeps the probes overhead under the defined constraints with an error of ±10%. Nonetheless, the relative error in the OWD estimate was not reported. In addition, detecting the delays of all the links takes around 50 to 80 seconds (see Fig. 7 in [69]).

5.3 SDN passive OWD measurement

Passive mechanisms for the OWD estimation in SDN rely on the modification of data packets to carry a timestamp at one or more measurement points. Such a modification is qualified by Zseby *et al.* as a semi-active scheme [98]. For the sake of simplicity, we choose to classify it as a passive technique as it does not employ probes.

5.3.1 Inband Network Telemetry. Hira and Wobker give an example of end-to-end OWD measurement using the Inband Network Telemetry (INT) specification, which is part of the P4 switch programming language [38]. The measurement is carried by OVS switches instances deployed at the end hosts. The first switch embeds in the data packets P4 instructions for the network nodes to report their forwarding latencies. Each intermediate switch adds in every packet a record of four bytes containing the time spent by this packet between its ingress and egress ports. The last switch collects these records of one hop latencies and adds them to calculate the packet end-to-end OWD.

INT assumes that switch processing and queuing delays dominate the transmission and propagation latencies. This is true for datacenter networks, which then do not require clock synchronization, but not WAN. It has the advantage of by-passing the SDN control plane altogether. Additionally, INT can pinpoint the node(s) responsible for long delays as the latency of each host is available in every packet at the last switch. For the same price, it also provides path information, which is useful for many applications. Nonetheless, modifying every data packet at each hop and at high traffic rates requires hardware timestamping for and accurate assessment of the delay. Otherwise, it will increase switch processing time and therefore the forwarding latency. In addition, this technique incurs a bandwidth overhead which grows linearly with number of packets and the path length.

5.3.2 Probabilistic Inband Network Telemetry. Ben-Basat *et al.* propose a probabilistic scheme to limit the packet overhead of INT by removing the growing telemetry header from the packets [8]. Instead, PINT uses aggregation operations (*e.g.,* mean or median) to encode the telemetry values (*e.g.,* the OWD) efficiently within a fixed bit-budget on the packets. This bit-budget can be shared by many telemetry queries, which are compiled by a query engine into an execution

plan (*i.e.,* a probability distribution on the query set) that is notified to the switches. Each switch determines probabilistically the query to run on the current packet by applying a global hash function on the packet ID. In other words, PINT implements a distributed sampling process, so every packet carries a value from a uniformly chosen switch on the flow's path. Furthermore, the bits of the aggregation result can be compressed using value approximation functions (with additive, multiplicative or random error). With respect to the median delay, PINT trades the measurement accuracy for a reduction of the processing and bandwidth overhead. Its measurement error is inversely proportional to both the bit-budget size and the number of packets per flow.

5.3.3 Data Plane Timestamping. Mizrahi and Moses argue for the use of Data Plane Timestamping (DTP) in SDN with three use-cases, which include delay measurement [60]. DTP adds a timestamp to the header of every packet when it traverses the first switch. It removes this stamp in the last switch before the packets leave the network. For the OWD estimation between any two switches, DTP requires the synchronization of their clocks. It divides the traffic into blocks of fixed time intervals (*e.g.,* one minute). For every block, the first switch saves the transmission time of the first packet, whereas the second switch keeps this packet's reception time. The DTP controller queries these saved times periodically and uses them to calculate the OWD at the path aggregate level.

DTP has a high overhead in terms of storage and processing, because of the stamping of every data packet. This needs to be done by hardware to keep up with high line rates. The authors evaluated DTP using the Emulab test-bed with 48 software-based switches. However, they did not report any delay measurement results. One might argue that the division of traffic into one minute intervals is a way of sampling, which alleviates the burden on the memory and processor of the measurement points and the controller. Nonetheless, with such intervals DTP will miss short latency spikes.

5.3.4 AM-PM. In RFC 8321 [30], Fioccola *et al.* suggest the Alternate Marking technique for Performance Monitoring (AM-PM), which improves the DTP mechanism described above. AM-PM replaces the DTP timestamps in the packet header with a single bit used as a mark (a color, 0 or 1), which distinguishes consecutive packet blocks. The timestamps are then saved by the network devices and collected by the management system. Nonetheless, even if AM-RM reduces the overhead of DTP it is vulnerable to packet loss and reordering.

Riesenberg *et al.* evaluated two implementations of AM-PM (the Marvell Prestera hardware switch and the reference P4 software switch bmv2) [70]. Their implementations use: PTP to synchronize the switches clocks, one second intervals to alternate the marking, one bit from the DSCP field of the IPv4 header as the marking bit, one register per switch to save the marking bit of the previous packet and one match-action rule that compares this register with the current packet bit. If the latter is toggled, the switch sends the current packet timestamp to the collector which calculates the OWD. The absolute error in measurement of hundreds of microseconds and few seconds delays is of 0.1 μs and 1 ms, respectively for the hardware and software implementations.

5.3.5 Marple. Narayana *et al.* propose the Marple query language to express flexible monitoring questions over the packet performance metadata captured by programmable switches [63]. This metadata (called packet performance stream) consists at each queue of one tuple per packet that includes the enqueue and dequeue timestamps. The queries can manipulate the tuples using new switch primitives to filter the packets (`filter`), modify their fields (`map`), aggregate information across packet (`groupby`), merge the queries results (`zip`) and stream them towards a collector server. For instance, the moving average of the queuing delay can be measured per flow with the `groupby` instruction, which takes the header fields identifying this flow as parameters. To realize this operation inside the switch at line rate, Marple uses a key-value store split into: a fast on-chip

cache in SRAM and an off-chip backing store in DRAM. The keys are the flow identifiers and the values are the state computed by the aggregation function.

The Marple key-value store organization has the advantage of limiting the memory requirement of performance measurement. Additionally, when combined with the `zip` instruction, it reduces the bandwidth and collector processing overhead in the case of streaming telemetry. Nevertheless, Marple cannot measure the delay between two nodes as it rejects the queries that necessitate the aggregation of information from multiple switches. This would require to stream all the packets to a central collector or coordinate between the switches which is hard to implement at line-rate.

5.3.6 P8. Harkous *et al.* carried a thorough testing of three P4 targets (Netronome SmartNIC, NetFPGA-SUM board and T4P4S DPDK-based software switch), which they programmed with various packet processing pipelines [35]. From the test results, the authors derived and validated a method for the prediction of P4 packet processing performance (P8). They found that the processing delay depends linearly on the number of: parsed, copied, modified, added and removed packet headers; and the number of match-action tables. They also concluded that the number of installed flow rules has a negligible impact on the measured delay. P8 presents the advantage of predicting a device packet processing delay just from analyzing its P4 programs. However, it requires the prior profiling of every P4 device as the processing delay specifically depends on the software/hardware implementation of the P4 constructs.

5.4 Comparison and Discussion of the SDN techniques

On the one hand, the majority of the existing OWD measurement mechanisms in SDN are based on OpenFlow. This is most likely due to the fact that OpenFlow is the most common southbound API. The PISA based mechanisms come next as P4 is reaching maturity. However, we have not found any publications on OWD measurement using NETCONF/YANG, probably because this framework's focus has primarily been configuration, though it can be extended to support probe injection and notifications of packet transit times. The same remark holds for gNMI/OpenConfig framework which, by contrast to the previous one, supports RTT measurement and focuses also (up from the start) on network operational state retrieval and streaming telemetry.

On the other hand, many of the OpenFlow OWD measurement mechanisms are active ones. This is probably due to the fact that most of the passive techniques need a hardware modification, for instance, to timestamp all data packets at line rate. By contrast, the active techniques do not require neither hardware modifications, nor software upgrades, thanks to the flexibility of OpenFlow. This facilitates their deployment in existing SDN infrastructure.

Tables 8, 9, 10, 11, 12, 13 and 14 recapitulate and compare all the SDN delay measurement mechanisms above. As all the active ones consider the total delay, we omit the components column from them. We also leave out the accuracy of passive techniques as, except the case of AM-PM, neither the magnitude of the delay nor its measurement error were reported in the reviewed works.

We do not include a clock synchronization column for the active class. We also strip the timestamping column from all the techniques. On the one hand, most of OWD measurement mechanisms in SDN do not need synchronization as they use the local clock of the controller or the destination switch to compare timestamps. On the other hand, none of the reviewed active measurement methods reported where timestamping takes place. Nonetheless, we think it is carried in user space if handled by the controller and in-kernel if managed by the destination switch, as done in TLL.

We do not use a separate column for the control bitrate used by the active techniques because all put the controller in the loop. Hence, their overhead on both planes (control and data) is determined approximately likewise by the probing rate and the probes size. The MDL and the TTL based

Table 8. The test setup of OWD active measurement techniques in SDN

Technique	Network (speed)	Cover.	Controller	Switches	Rules/switch
CL	Mininet (n/r)	link	FloodLight	20	0
CLL	Mininet (n/r)	link	Nox	30	0
MDL	Testbed (n/r)	path	POX	3	F(I,n)
OpenNetMon	Testbed (100 Mbps)	path	POX	4	n/r
TL	Mininet (100 Mbps)	link	n/r	4	3
TLL	Mininet (100 Mbps)	link	Nox C	60	0 (hard-coded)
SLAM	Testbed (100 Mbps)	path	POX	12	n/r
SDProber	Mininet (n/r)	path	RYU	196	p+4

Legend: $F()$=linear function, I=# iterations, n=# switches, p=# switch ports

Table 9. The probing in OWD active measurement techniques in SDN

Technique	Process	Rate	Size (bytes)*
CL	Fixed	2 pps per link	24
CLL	~	~	40
MDL	Fixed	n/r	n/r
OpenNetMon	Adaptive	F(throughput)	72
TL	Fixed	n/r	40
TLL	Fixed	1–100 pps per link	44
SLAM	Fixed	1 pps per path	n/r
SDProber	Adaptive	20 to 60 ppm per link	n/r

Legend: *=Ethernet frame size as reported (even if < 64 bytes), pps=packet per second, ppm=packet per minute

Table 10. The accuracy and time complexity of OWD active measurement techniques in SDN

	Reported Accuracy		
Technique	**Tested Delay**	**Error**	**Time Complexity**
CL	0–20 ms	1%	$O(1)$
CLL	5 ms	3%	$O(1)$
MDL	540 μs RTT	3.7%	$O(1)$ one path $O((n+e)(c+1))$ all links
OpenNetMon	7 ms	2.3%	$O(1)$
TL	0–350 ms	n/r	$O(1)$
TLL	1, 5, 10 ms RTT	3, 0.4, 0.1%	$O(1)$ one link $O(Kn)$ all links
SLAM	0–20 ms	n/r	$O(N)$
SDProber	n/r	n/r	$O(n^3)$ all links

Legend: n=# switches, e=# links, c=# simple loops, N=# probes, K=maximum size of the vertex cover

Table 11. The test setup of OWD passive measurement techniques in SDN

Technique	Network (links speed)	Controller	Switches
INT	n/r	not needed	n/r
PINT	NS3 (100 Gbps)	query engine	n/r
DPT	Emulab (100 Mbps)	dpctl	OVS
AM-PM	Testbed & Mininet (10 Gbps)	collector	Marvell Prestera & bmv2
Marple	Testbed (100 Gbps)	not needed	Marple targets
P8	Testbed (10 Gbps)	not needed	different P4 targets

Table 12. The delay structure of OWD passive measurement techniques in SDN

Technique	Coverage	Components	Granularity	Sampling rate
INT	end-to-end	total	per-packet	100%
PINT	end-to-end	total	per-flow	1/q
DPT	per path	total	aggregate	1 ppm
AM-PM	per path	total	aggregate	1 pps
Marple	per hop	queuing	per-packet	100%
P8	per hop	processing	per-packet	100%

Legend: q=# queries

Table 13. The state used by OWD passive measurement techniques in SDN

Technique	Data structure
INT	H latency records per packet (H is the # of hops)
PINT	User fixed bit-budget per packet
DPT	Two timestamps per minute per path
AM-PM	One register + Two timestamps per second per path
Marple	Two timestamps per packet per queue
P8	Profile-vector per P4 target + Features-vector per P4 program

techniques constitute an exception. Their data plane bandwidth needs to be multiplied by the number of their loop iterations.

With regard to robustness, none of the active techniques addressed the issues of packet loss, reordering or duplication. Though, these effects can be tackled with the use of a sequence number as discussed above for traditional networks (Section 4.1.8). Conversely, the data packets timestamps used by the passive schemes allow the detection of all three issues.

Some of the active techniques showed that controller driven measurement is not suitable for delays in the sub-millisecond scale. This is due to the time uncertainties of the control plane, which is highly tributary of the controller performance, the control channel speed and the software scheduling of the switches. Overcoming this limit is achieved at a high cost, as it requires the increase of the probe looping in the data plane (*e.g.,* MDL). In this regard, the passive schemes based on P4 are better as they do not need a controller. They incur either a lower network overhead (*e.g.,* to in-band timestamps of INT) or none (*e.g.,* AM-PM and P8, if we put aside the initial device profiling).

Table 14. The cost of OWD passive measurement techniques in SDN

Technique	Control rate	Synchronization	Complexity
INT	0 (not needed)	not needed	$O(H)$
PINT	0 (not needed)	not needed	$O(1)$
DPT	n/r	n/r	$O(B)$
AM-PM	n/r	PTP	$O(B)$
Marple	$r(s, c, l)$	not needed	$O(n)$
P8	0 (not needed)	not needed	$O(1)$

Legend: n/r=not reported, H=# hops, B=# blocks, n=# packets
$r()$: rate of tuples evicted from the switch cache towards the collector server
s=monitored state size, c=cache size, l=data line rate

The SDN central global view of the network facilitates simultaneous delay measurements with a lower overhead compared to traditional networks. In the mechanisms that rely on a controller, the latter knows about the links shared between different paths and can therefore measure them once and share the result (*e.g.,* MDL and TLL). It can also adapt the probing rate based on the flows state and network operator requirements (*e.g.,* OpenNetMon and SDProber). However, in addition to being a single point of failure, the controller could be a performance bottleneck. So the rate of the measurement messages needs to be paced according to the control plane capacities. Likewise, the mechanisms that rely on a query engine benefit also from the network global view to plan the execution of multiple measurement queries with limited overhead. Though, this comes at the price of a reduced accuracy (*e.g.,* PINT) or a limited coverage (*e.g.,* Marple).

The planning of network-wide measurements with a minimum overhead requires in some cases an exponential search over the network topology. For instance, finding a set of all spanning trees in the many loops technique, a minimum vertex cover in the TTL based LLDP looping and a constraint satisfying set of shortest paths in SDProber. In addition, this search is inflexible as changes in the network may require a re-computation. Therefore, polynomial heuristic solutions like greedy algorithms and pseudo random walk are used instead.

In most of the active mechanisms above, the controller pre-configures the flow forwarding rules in the switches. So the OWD measurement is not affected by the rule setup latency caused by the *PacketIn–FlowMod* message exchange. However, in reactive SDN, the flow rule setup happens only after the arrival of the flow's first packet. This might be the case for the passive mechanisms above. Zhang and Liu showed that in this case the forwarding of a flow's first packet can be up to 28 times longer in WAN SDN than in traditional networks [95]. As a consequence, the measurement of the average delay of small flows will be skewed towards the delay of their first packet. This drawback could prevent the use of reactive SDN over WAN for many applications. By contrast, the P4 based techniques do not incur a flow setup as their match-action tables are pre-populated by the P4 compiler.

The techniques that compose a path OWD by summing the delays of its links have a common disadvantage. They dot not mimic exactly the transit of the data packets along the path. But they rather work as if the delays of the consecutive links were independent. Nguyen *et al.* showed that the variance of the delay measured this way can be three times smaller than the exact one [64]. The underestimation of the delay variance has a direct impact on the accuracy of the average delay estimation.

Finally, the overwhelming majority of reviewed SDN works are not capable of measuring end-to-end delay. This is due to the lack of information about the end hosts and their links with the edge switches. This limitation can be overcome by exposing SDN to the end hosts or by deploying virtual switches there as done by the INT method.

5.5 Comparison of OWD measurement between traditional networks and SDN

On the one hand, we note that some of the techniques about OWD measurement in traditional networks have been reused in SDN. First, the trivial RTT halving scheme has been reused in several SDN research works to estimate the delay between the controller and the switches (*i.e.,* the control channel latency); and between the switches themselves (*i.e.,* the latency inside the data plane). This scheme does not require time synchronization between or inside the different SDN planes. However, it adds a network and a processing overhead on the controller.

Second, the idea of measuring links delays through a cyclic-path has been reused in SDN by putting the controller in the loop. This loop includes also two or more switches. Hence, the delay between the latter is be measured by subtracting the control channel latency. This reuse has the same advantage and drawback as the previous one.

Third, the ideas based on NetFlow have inspired similar ones in SDN where a flow OWD is estimated by comparing the times of flow setup messages from a path's first and last switch. In this case, the slow path taken by the flow's first packet includes the round trip latency to the controller and the processing time of the latter. Both latencies need to be removed to obtain an accurate data plane delay.

On the other hand, we found that OWD measurement benefited from SDN in ways that are not possible in traditional networks. Particularly, the path taken by the test probes or the normal data packets in the traditional networks is determined by independent routers. So if the applications require this path, for instance to check for multipath routing, they need to use a separate tool like `traceroute`, which relies on ICMP. If the latter is routed differently compared to the probes or the data packets, there will be a discrepancy between the obtained delay and the reported test path. Conversely, SDN does not suffer from this issue as it has complete control, therefore knowledge, of the path taken by any packets.

From the previous observation we can draw the following consequences. First, when combined with the path information, SDN based delay measurement techniques are more accurate than traditional networks' schemes. Second, they are also more robust to dynamic route changes. Third, in some SDN techniques the controller can compile a path OWD by adding the delays of its consecutive links. In contrast, this is not possible in traditional networks due to the lack of path information, unless a separate path discovery tool is used (with the above risk of discrepancy).

More importantly, for the same reasons of central control and flexibility, SDN is better suited than traditional networks when it comes to running many measurements in parallel. The hosts in traditional networks might duplicate the measurements of some shared links or segments, increase the overhead on the network and influence each other results. By contrast, in SDN the controller or the query engine orchestrates network-wide measurements in a more efficient way. That is it selects optimal paths and nodes to minimize the collective overhead and still provides a better accuracy.

Finally, from the type of test networks and traffic traces used by the reviewed mechanisms we observe that: if we put a aside the active measurement techniques in traditional networks (all but RLI), most of the remaining ones have been evaluated at 10 Gbps bandwidth. So they are suitable for use in high speed networks such as datacenter ones, where SDN is mostly deployed. Furthermore, the traditional measurement techniques (both active and passive) have been validated in WAN for over a decade, whereas we have not find any publications that address OWD measurement in

SD-WAN. One might argue that the controller driven schemes will suffer a decrease in accuracy due to the control channel high latency. This makes the SDN mechanisms that do not require s controller, such as the previous P4 based solutions, good candidates for SD-WAN deployment.

6 CONCLUSION AND FUTURE WORK

We set out to survey the topic of OWD measurement in traditional IP networks and in SDN. We have covered a representative set of standards and research works. Overall, we summarized each work in a way that makes its key ideas easily understandable without overwhelming details. We discussed the merits and the limitations of each one individually and collectively. Using rich table formats, we compared active and passive techniques according to important usage criteria: applicability, accuracy, cost and robustness.

An important lesson we learned from this survey is that it is crucial to the choose the right mechanism based on its usage criteria. Many of these mechanisms have parameters (*e.g.,* probing and sampling rates) that need careful tuning to balance the network overhead and the desired accuracy.

We also discussed the reuse in SDN of some techniques developed for traditional networks. Another lesson we can draw here is about the benefit brought by SDN. Namely, it allows network wide delay measurements with a lower overhead and a better accuracy, thanks to its flexibility, central control and the use of polynomial heuristics to avoid the price of an exponential search over the network topology.

Unlike P4 based techniques, the controller driven network-wide measurements still pose the risk of transforming the controller into a performance bottleneck in the case of large throughput and high number of flows. A possible solution to this main challenge could be the distribution of the overhead over many physical controllers. This would require some form of planning and coordination, which we leave for future research. Likewise, the use of controller driven measurements to assess sub-millisecond delays without high probing frequencies remains an open question.

For special use-cases, like real time applications, we think that the bench-marking of few candidate techniques might be required before picking the most adequate. Inspired by `CBench` [86], `OFLOPS` [72] and `Whippersnapper` [22], which are performance benchmarks for, respectively, SDN controllers, OpenFlow switches and P4 compilers, we advocate for the development of a common open benchmark for testing delay measurement mechanisms. This would also benefit the developers of new techniques and make the comparison with the existing ones easier.

As a possible future work, we believe that combining OWAMP with OpenFlow will lead to a wide deployment of the former as both protocols are mature standards. This idea is easy to realize as the logical components of OWAMP fit in the SDN architecture very well. Namely, the Sender and the Receiver can be implemented in the switches, which then need to synchronize their clocks and implement the Poisson paced probing. Alternatively, to avoid this switch modification, the controller can drive the Poisson process for the price of an increased message exchange with the data plane. Additionally, the OWAMP Control and Fetch clients can be considered as applications that run on top of the SDN controller. Furthermore, the OWAMP Server can be naturally implemented by the SDN controller and its messages can be added to OpenFlow as an extension. Nevertheless, compared to `owping/owampd`, the combination of OWAMP and OpenFlow adds the communication latency of the control channel. In the traditional OWAMP implementation, the Server is hosted with the Receiver, while the Control and Fetch clients co-exist with the Sender.

Concerning the measurement techniques based on aggregate data structures (*e.g.,* LDA, FineComb, LDS, etc.), if we put aside: (i) the difficulty of transforming them standards, (ii) the resistance of major router vendors to implement them and (iii) the reluctance of Internet Service Providers (ISPs) to invest in them; one can easily imagine their integration into SDN switches. The aggregate data

structures can be implemented in the ternary content-addressable memory (TCAM) as extensions of the switches' flow tables, though with an extra hardware cost. They might also profit from the flows information that already exists there (*e.g.,* byte counters) to adapt the probing or the sampling rate.

With respect to network query languages, we found that only MAPLE and Marple support a time dependent metric such as the OWD, so far. In particular, the compact query interface devised for MAPLE can be used by the controller as an OpenFlow extension for statistics collection with little overhead. In general, it would be interesting the extend the expressiveness of other recent query languages (*e.g.,* PathQuery, Sonata and NetQRE) with instructions that support the OWD measurement.

Regarding other future works, the cyclic-path OWD measurement techniques can be used with Multipath TCP for delay based packet scheduling and congestion control. Furthermore, we have not found research works about OWD measurement in hybrid networks, which are composed by connecting traditional IP networks and software defined ones. So the issue of measuring this metric in such configurations remains open.

Finally, most of the schemes we surveyed did not address the issue of security. One exception is OWAMP which uses authentication and encryption to prevent theft of service and tampering with the results. The identification of other threats, either general or specific to a certain technique, would be a valuable addition. Moreover, a rigorous security analysis of the surveyed techniques against identified threats is essential for a safe and large scale deployment.

ACKNOWLEDGMENTS

The author would like to thank the anonymous reviewers for their constructive feedback.

REFERENCES

[1] Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. 2016. Research Challenges for Traffic Engineering in Software Defined Networks. *IEEE Network* 30, 3 (2016), 52–58.

[2] Rasool Al-Saadi, Grenville Armitage, Jason But, and Philip Branch. 2019. A Survey of Delay-Based and Hybrid TCP Congestion Control Algorithms. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3609–3638.

[3] Guy Almes, Sunil Kalidindi, Matthew Zekauskas, and Al Morton. 2016. A One-Way Delay Metric for IP Performance Metrics (IPPM). RFC 7679. https://doi.org/10.17487/RFC7679

[4] V Altukhov and EEvgeny Chemeritskiy. 2014. On Real-Time Delay Monitoring in Software-Defined Networks. In *IEEE MoNeTeC*. 1–6.

[5] Brice Augustin, Timur Friedman, and Renata Teixeira. 2010. Measuring Multipath Routing in the Internet. *IEEE/ACM Transactions on Networking* 19, 3 (2010), 830–840.

[6] Francois Baccelli, Sridhar Machiraju, Darryl Veitch, and Jean C Bolot. 2007. On Optimal Probing for Delay and Loss Measurement. In *ACM IMC*. 291–302.

[7] Vaibhav Bajpai and Jürgen Schönwälder. 2015. A Survey on Internet Performance Measurement Platforms and Related Standardization Efforts. *IEEE Communications Surveys & Tutorials* 17, 3 (2015), 1313–1341.

[8] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: probabilistic in-band network telemetry. In *ACM SIGCOMM Conference*. 662–680.

[9] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. 2014. ONOS: towards an open, distributed SDN OS. In *ACM HotSDN*.

[10] Martin Björklund. 2010. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020. https://doi.org/10.17487/RFC6020

[11] Burton H Bloom. 1970. Space/time Trade-Offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (1970), 422–426.

[12] Jean-Chrysotome Bolot. 1993. End-to-End Packet Delay and Loss Behavior in the Internet. In *ACM SIGCOMM Conference*. 289–298.

[13] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.

[14] CJ Bovy, HT Mertodimedjo, Gerard Hooghiemstra, Henk Uijterwaal, and Piet Van Mieghem. 2002. Analysis of End-to-End Delay Measurements in Internet. In *PAM Conference*.
[15] Djalel Chefrour. 2021. Evolution of Network Time Synchronization Towards Nanoseconds Accuracy: A Survey. Manuscript submitted for publication.
[16] Stuart Cheshire. 1996. Latency and the Quest for Interactivity. In *White paper commissioned by Volpe Welty Asset Management, LLC, for the Synchronous Person-to-Person Interactive Computing Environments Meeting.*
[17] Jin-Hee Choi and Chuck Yoo. 2005. One-Way Delay Estimation and Its Application. *Computer Communications* 28, 7 (2005), 819–828.
[18] Cisco. 2012. Introduction to Cisco IOS NetFlow - A Technical Overview. White Paper, Last updated: May 2012.
[19] Cisco. 2018. *IP SLAs Overview.* Retrieved January 05, 2021 from https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipsla/configuration/15-s/sla-15-s-book/sla_overview.html
[20] Alexander Clemm and Eric Voit. 2019. Subscription to YANG Notifications for Datastore Updates. RFC 8641. https://doi.org/10.17487/RFC8641
[21] Corvil. 2020. *Corvil Appliances.* Retrieved June 30, 2020 from https://www.pico.net/corvil-analytics/corvil-classic/corvil-appliances
[22] Huynh Tu Dang, Han Wang, Theo Jepsen, Gordon Brebner, Changhoon Kim, Jennifer Rexford, Robert Soulé, and Hakim Weatherspoon. 2017. Whippersnapper: A P4 language benchmark suite. In *ACM SOSR*. 95–101.
[23] Luca De Vito, Sergio Rapuano, and Laura Tomaciello. 2008. One-Way Delay Measurement: State of the Art. *IEEE Transactions on Instrumentation and Measurement* 57, 12 (2008), 2742–2750.
[24] Stephen Donnelly, Ian Graham, and René Wilhelm. 2001. Passive Calibration of an Active Measurement System. In *PAM Conference*. Springer-Verlag, Berlin, 1–8.
[25] John Eidson and Kang Lee. 2002. IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. In *2nd ISA/IEEE Sensors for Industry Conference*, Vol. 10. 98–105.
[26] Rob Enns, Martin Björklund, Andy Bierman, and Jürgen Schönwälder. 2011. Network Configuration Protocol (NETCONF). RFC 6241. https://doi.org/10.17487/RFC6241
[27] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2013. The Road to SDN. *ACM Queue* 11, 12 (2013), 20–40.
[28] Yu Feng, Yue Pan, Bo Zhu, Yuchuan Deng, Jing Wu, and Hao Jiang. 2019. A New Framework for Network Flow Queuing Delay Prediction Based on Stream Computing. In *IEEE 5th BigDataSecurity, HPSC and IDS Conference*. 212–217.
[29] Paolo Ferrari, Alessandra Flammini, Emiliano Sisinni, Stefano Rinaldi, Dennis Brandão, and Murilo Silveira Rocha. 2018. Delay Estimation of Industrial IoT Applications Based on Messaging Protocols. *IEEE Transactions on Instrumentation and Measurement* 67 (2018), 2188–2199.
[30] Giuseppe Fioccola, Alessandro Capello, Mauro Cociglio, Luca Castaldelli, Mach Chen, Lianshu Zheng, Greg Mirsky, and Tal Mizrahi. 2018. Alternate-Marking Method for Passive and Hybrid Performance Monitoring. RFC 8321. https://doi.org/10.17487/RFC8321
[31] Jim Gettys and Kathleen Nichols. 2011. Bufferbloat: Dark buffers in the Internet. *ACM Queue* 9, 11 (2011), 40–54.
[32] Ian D. Graham, Stephen Donnelly, Julie Martin, and John G. Cleary. 1998. Nonintrusive and Accurate Measurement of Unidirectional Delay and Delay Variation on the Internet. In *INET'98*. Internet Society.
[33] Omer Gurewitz, Israel Cidon, and Moshe Sidi. 2006. One-Way Delay Estimation Using Network-Wide Measurements. *IEEE Transactions on Information Theory* 52, 6 (2006), 2710–2724.
[34] Taimur Hafeez, Nadeem Ahmed, Bilal Ahmed, and Asad Waqar Malik. 2017. Detection and Mitigation of Congestion in SDN Enabled Data Center Networks: A Survey. *IEEE Access* 6 (2017), 1730–1740.
[35] Hasanin Harkous, Michael Jarschel, Mu He, Rastin Pries, and Wolfgang Kellerer. 2020. P8: P4 with Predictable Packet Processing Performance. *IEEE Transactions on Network and Service Management* (2020).
[36] Stephen Hemminger et al. 2005. Network Emulation with NetEm. In *The 6th LCA*.
[37] Christian Henke, Carsten Schmoll, and Tanja Zseby. 2008. Empirical Evaluation of Hash Functions for Multipoint Measurements. *ACM SIGCOMM Computer Communication Review* 38, 3 (2008), 39–50.
[38] Mukesh Hira and LJ Wobker. 2015. *Improving Network Monitoring and Management with Programmable Data Planes.* Retrieved June 27, 2020 from https://p4.org/p4/inband-network-telemetry/
[39] Internet2. 2016. *One-Way Ping (OWAMP).* Retrieved June 12, 2020 from http://software.internet2.edu/owamp/
[40] Manar Jammal, Taranpreet Singh, Abdallah Shami, Rasool Asal, and Yiming Li. 2014. Software Defined Networking: State of the Art and Research Challenges. *Computer Networks* 72 (2014), 74–98.
[41] Sunil Kalidindi, Matthew Zekauskas, and Guy Almes. 1999. A One-way Delay Metric for IPPM. RFC 2679. https://doi.org/10.17487/RFC2679
[42] Sunil Kalidindi and Matthew J Zekauskas. 1999. Surveyor: An Infrastructure for Internet Performance Measurements. In *INET'99*. Internet Society.
[43] Murat Karakus and Arjan Durresi. 2017. Quality of Service (QoS) in Software Defined Networking (SDN): A survey. *Journal of Network and Computer Applications* 80 (2017), 200–218.

[44] Jochen Kögel. 2011. One-Way Delay Measurement Based on Flow Data: Quantification and Compensation of Errors by Exporter Profiling. In *IEEE ICOIN*. 25–30.
[45] Ramana Kompella, Kirill Levchenko, Alex C Snoeren, and George Varghese. 2009. Every Microsecond Counts: Tracking Fine-Grain Latencies with a Lossy Difference Aggregator. *ACM SIGCOMM Computer Communication Review* 39, 4 (2009), 255–266.
[46] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2014. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* 103, 1 (2014), 14–76.
[47] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *ACM SIGCOMM Conference*. 514–528.
[48] James F. Kurose and Keith W. Ross. 2012. *Computer Networking: A Top-Down Approach* (6th ed.). Pearson, 42.
[49] Myungjin Lee, Nick Duffield, and Ramana ella. 2012. MAPLE: A Scalable Architecture for Maintaining Packet Latency Measurements. In *ACM IMC*. 101–114.
[50] Myungjin Lee, Nick Duffield, and Ramana Kompella. 2010. Not All Microseconds Are Equal: Fine-Grained Per-Flow Measurements with Reference Latency Interpolation. In *ACM SIGCOMM Conference*. 27–38.
[51] Myungjin Lee, Nick Duffield, and Ramana Kompella. 2010. Two Samples Are Enough: Opportunistic Flow-Level Latency Estimation Using NetFlow. In *IEEE INFOCOM*. 1–9.
[52] Myungjin Lee, Sharon Goldberg, Ramana Kompella, and George Varghese. 2011. Fine-Grained Latency and Loss Measurements in the Presence of Reordering. In *ACM SIGMETRICS Conference*. San Jose, CA, USA, 329–340.
[53] Young Lee, Ricard Vilalta, Ramon Casellas, Ricardo Martínez, and Raul Muñoz. 2017. Scalable telemetry and network autonomics in ACTN SDN controller hierarchy. In *IEEE 19th ICTON*. 1–4.
[54] Lingxia Liao and Victor CM Leung. 2016. LLDP Based Link Latency Monitoring in Software Defined Networks. In *IEEE CNSM*. 330–335.
[55] Lingxia Liao, Victor CM Leung, and Min Chen. 2018. An Efficient and Accurate Link Latency Monitoring Method for Low-Latency Software-Defined Networks. *IEEE Transactions on Instrumentation and Measurement* 68, 2 (2018), 377–391.
[56] László Lovász. 1993. Random Walks on Graphs: A Survey. *Combinatorics, Paul erdos is eighty* 2, 1 (1993), 1–46.
[57] Wei-Zhou Lu, Wei-Xuan Gu, and Shun-Zheng Yu. 2009. One-Way Queuing Delay Measurement and Its Application on Detecting DDoS Attack. *Journal of Network and Computer Applications* 32, 2 (2009), 367–376.
[58] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. 2014. Opendaylight: Towards a model-driven sdn controller architecture. In *IEEE WoWMoM*. 1–6.
[59] Mininet Team. 2018. *Mininet: An Instant Virtual Network on your Laptop (or other PC).* Retrieved July 14, 2020 from http://mininet.org/
[60] Tal Mizrahi and Yoram Moses. 2016. *The Case for Data Plane Timestamping in SDN.* Technical Report. Technion. http://arxiv.org/pdf/1602.03342v1
[61] Tal Mizrahi and Yoram Moses. 2016. Time Capability in NETCONF. RFC 7758. https://doi.org/10.17487/RFC7758
[62] Maurizio Molina, Fredric Raspall, Saverio Niccolini, Nick Duffield, and Tanja Zseby. 2009. Sampling and Filtering Techniques for IP Packet Selection. RFC 5475. https://doi.org/10.17487/RFC5475
[63] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *ACM SIGCOMM Conference*. 85–98.
[64] Huu-Nghi Nguyen, Thomas Begin, Anthony Busson, and Isabelle Guérin Lassous. 2016. Evaluation of an End-to-End Delay Estimation in the Case of Multiple Flows in SDN Networks. In *IEEE CNSM*. 336–341.
[65] Kathleen Nichols and Van Jacobson. 2012. Controlling Queue Delay. *Commun. ACM* 55, 7 (2012), 42–50.
[66] Anne-Cécile Orgerie, Paulo Gonçalves, Matthieu Imbert, Julien Ridoux, and Darryl Veitch. 2012. Survey of Network Metrology Platforms. In *IEEE/IPSJ 12th SAINT*. Izmir, Turkey, 220–225.
[67] Abhinav Pathak, Himabindu Pucha, Ying Zhang, Y Charlie Hu, and Z Morley Mao. 2008. A Measurement Study of Internet Delay Asymmetry. In *PAM Conference*. Springer-Verlag, Berlin, 182–191.
[68] Kevin Phemius and Mathieu Bouet. 2013. Monitoring Latency with OpenFlow. In *IEEE CNSM*. 122–125.
[69] Sivaramakrishnan Ramanathan, Yaron Kanza, and Balachander Krishnamurthy. 2018. SDProber: A Software Defined Prober for SDN. In *ACM SOSR*. 1–7.
[70] Alon Riesenberg, Yonnie Kirzon, Michael Bunin, Elad Galili, Gidi Navon, and Tal Mizrahi. 2019. Time-multiplexed parsing in marking-based network telemetry. In *ACM SYSTOR*. 80–85.
[71] Elisa Rojas, Roberto Doriguzzi-Corin, Sergio Tamurejo, Andres Beato, Arne Schwabe, Kevin Phemius, and Carmen Guerrero. 2018. Are we ready to drive software-defined networks? A comprehensive survey on management tools and techniques. *ACM Computing Surveys (CSUR)* 51, 2 (2018), 1–35.

[72] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W Moore. 2012. OFLOPS: An open framework for OpenFlow switch evaluation. In *PAM Conference*. Springer, 85–95.
[73] Josep Sanjuàs-Cuxart, Pere Barlet-Ros, Nick G. Duffield, and Ramana Kompella. 2011. Sketching the Delay: Tracking Temporally Uncorrelated Flow-Level Latencies. In *ACM IMC*.
[74] Yaron Schwartz, Yuval Shavitt, and Udi Weinsberg. 2010. A Measurement Study of the Origins of End-to-End Delay Variations. In *PAM Conference*. Springer-Verlag, Berlin, 21–30.
[75] Muhammad Shahzad and Alex X Liu. 2014. Noise Can Help: Accurate and Efficient Per-Flow Latency Measurement without Packet Probing and Time Stamping. *ACM SIGMETRICS Conference* 42, 1 (2014), 207–219.
[76] Stanislav Shalunov, Greg Hazel, Jana Iyengar, and Mirja Kühlewind. 2012. Low Extra Delay Background Transport (LEDBAT). RFC 6817. https://doi.org/10.17487/RFC6817
[77] Minsu Shin, Mankyu Park, Deockgil Oh, Byungchul Kim, and Jaeyong Lee. 2011. Clock Synchronization for One-Way Delay Measurement: A Survey. In *International Conference on Advanced Communication and Networking*. Springer-Verlag, Berlin, 1–10.
[78] Zhaogang Shu, Jiafu Wan, Jiaxiang Lin, Shiyong Wang, Di Li, Seungmin Rho, and Changcai Yang. 2016. Traffic Engineering in Software-Defined Networking: Measurement and Management. *IEEE Access* 4 (2016), 3246–3256.
[79] Debanshu Sinha, K Haribabu, and Sundar Balasubramaniam. 2015. Real-Time Monitoring of Network Latency in Software Defined Networks. In *IEEE ANTS Conference*. 1–3.
[80] Peerapon Siripongwutikorn and Sujata Banerjee. 2002. Per-Flow Delay Performance in Traffic Aggregates. In *IEEE GLOBECOM*, Vol. 3. 2634–2638.
[81] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. 2007. Accurate and Efficient SLA Compliance Monitoring. In *ACM SIGCOMM Conference*. 109–120.
[82] Philipp Svoboda, Markus Laner, Joachim Fabini, Markus Rupp, and Fabio Ricciato. 2012. Packet Delay Measurements in Reactive IP Networks. *IEEE Instrumentation & Measurement Magazine* 15, 6 (2012), 36–44.
[83] The Open Networking Foundation. 2015. OpenFlow Switch Specification, Version 1.5.1. Retrieved July 2, 2020 from https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
[84] The OpenConfig operators group. 2015. OpenConfig repository. Retrieved January 8, 2021 from https://github.com/openconfig/public
[85] The OpenConfig operators group. 2018. gNMI - gRPC Network Management Interface, Version: 0.6.0. Retrieved January 8, 2021 from https://github.com/openconfig/reference/tree/master/rpc/gnmi
[86] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. 2012. On Controller Performance in Software-Defined Networks. In *2nd USENIX Hot-ICE*.
[87] Pang-Wei Tsai, Chun-Wei Tsai, Chia-Wei Hsu, and Chu-Sing Yang. 2018. Network Monitoring in Software-Defined Networking: A Review. *IEEE Systems Journal* 12, 4 (2018), 3958–3969.
[88] Ahmad Vakili and Jean-Charles Gregoire. 2012. Accurate One-Way Delay Estimation: Limitations and Improvements. *IEEE Transactions on Instrumentation and Measurement* 61, 9 (2012), 2428–2435.
[89] Niels LM Van Adrichem, Christian Doerr, and Fernando A Kuipers. 2014. OpenNetNon: Network Monitoring in OpenFlow Software-Defined Networks. In *IEEE NOMS*. 1–8.
[90] Darryl Veitch, Julien Ridoux, and Satish Babu Korada. 2008. Robust Synchronization of Absolute and Difference Clocks Over Networks. *IEEE/ACM Transactions on Networking* 17, 2 (2008), 417–430.
[91] Junfeng Wang, Mingtian Zhou, and Yuxia Li. 2004. Survey on the End-to-End Internet Delay Measurements. In *HSNMC*. Springer-Verlag, Berlin, 155–166.
[92] Abdulsalam Yassine, Hesam Rahimi, and Shervin Shirmohammadi. 2015. Software Defined Network Traffic Measurement: Current Trends and Challenges. *IEEE Instrumentation & Measurement Magazine* 18, 2 (2015), 42–50.
[93] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V Madhyastha. 2015. Software-Defined Latency Monitoring in Data Center Networks. In *PAM Conference*. Springer-Verlag, Berlin, 360–372.
[94] Matthew Zekauskas, Anatoly Karp, Stanislav Shalunov, Jeff Boote, and Benjamin Teitelbaum. 2006. A One-way Active Measurement Protocol (OWAMP). RFC 4656. https://doi.org/10.17487/RFC4656
[95] Ting Zhang and Bin Liu. 2019. Exposing End-to-End Delay in Software-Defined Networking. *International Journal of Reconfigurable Computing* 2019 (2019).
[96] Yusu Zhao, Pengfei Zhang, and Yaohui Jin. 2016. Netography: Troubleshoot your network with packet behavior in SDN. In *IEEE NOMS*. 878–882.
[97] Tanja Zseby and Florian Schreiner. 2002. *QoS Monitoring and Measurement Benchmarking*. Research Report. Fraunhofer FOKUS.
[98] Tanja Zseby, Sebastian Zander, and Georg Carle. 2001. Evaluation of Building Blocks for Passive One-Way-Delay Measurements. In *PAM Conference*.