SDProber: A Software Defined Prober for SDN

Sivaramakrishnan Ramanathan*
University of Southern California
satyaman@usc.edu

Yaron Kanza
AT&T Labs-Research
kanza@research.att.com

Balachander Krishnamurthy
AT&T Labs-Research
bala@research.att.com

ABSTRACT

Proactive measurement of the delay in communication networks aims to detect congestion as early as possible and find links on which the traffic flow is obstructed. There is, however, a tradeoff between the detection time and the cost (e.g., bandwidth utilization). An *adaptive measurement* adjusts the inspection rate per each link, for effective monitoring with reduced costs. In this paper we show how adaptive measurement can be implemented effectively in SDN. We present SDProber—a tool for proactive measurement of delays in SDN. SDProber uses *probe packets* that are routed by adding tailored rules to the vSwitches. It adjusts the forwarding rules to route probe packets more frequently to areas where congestion tends to occur. To increase efficiency, instead of computing complex routes for probe packets, SDProber uses a novel approach of probing by a random walk. Adaptation is achieved by changing the probabilities that govern the random walk. Our experimental results show that SDProber provides control over the probe rates per each link and that it reduces measurement costs in comparison to baseline methods that send probe packets via shortest paths.

CCS CONCEPTS

• **Networks** → **Network measurement**; *Network performance analysis*; *Network reliability*;

KEYWORDS

SDN, measurement, probe packets, delay, random walk, OpenFlow

*Work done while at AT&T Labs-Research.

SOSR '18, March 28–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5664-0/18/03…$15.00
https://doi.org/10.1145/3185467.3185472

ACM Reference Format:
Sivaramakrishnan Ramanathan, Yaron Kanza, and Balachander Krishnamurthy. 2018. SDProber: A Software Defined Prober for SDN. In *SOSR '18: ACM SIGCOMM Symposium on SDN Research, March 28–29, 2018, Los Angeles, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3185467.3185472

1 INTRODUCTION

Measurements are crucial for managing communication networks, e.g., for troubleshooting and to maintain service-level agreements (SLAs). However, measurements incur costs in the form of bandwidth, CPU and memory utilization. In this paper we show how the central control in SDN can be used for reducing the costs that are involved in proactive delay measurements, and how SDN can facilitate adaptability of the measurements to varying conditions.

Persistent delays in wide area networks are often perilous and can adversely affect the effectiveness of online commerce in the stock exchange, streaming video and audio, online games, and other online applications, where obstructed data transfer can cause a significant loss of money or disrupt the quality of service. Thus, it is essential to *proactively* detect long delays as early as possible, and cope with the hindrance, as soon as it starts.

Delays are detected by periodically inspecting the links of the network. However, there is a tradeoff between the detection time and the cost. Increasing the inspection rate of a link can reduce the detection time of a delay, while inspecting a link too often could hinder traffic via that links or via the nodes it connects. It is, thus, desirable to limit the inspection rate per each link. A lower bound would specify how often the link should be inspected, to prevent a congestion that is unobserved or detected late. An upper bound would restrict the number of inspections per link, so that the measurement would not obstruct network traffic.

Network congestion events are frequent and are distributed unevenly. The frequency of congestion and high delays, thus, could be learned, and the inspection rates would be modified accordingly. Links that were not inspected for a long time and frequently-delayed links would receive a high priority when considering which links to examine next. The goal is to inspect each link at a rate that is within the specified limits. Traditional tools, like *ping* and *traceroute*, however, are unsuitable for adaptive measurement, where different links

should be inspected at different rates. To illustrate this, consider a network where the links are arbitrarily partitioned into two groups. The links of one group should be probed at a rate of x inspections per minute, and the others, at a rate of $2x$. Such constraints often cannot be satisfied when measuring round-trip times via predefined paths, e.g., when links from the two groups complete the round trip of one another. SDN's central control over forwarding rules, however, allows for efficient implementation of adaptable delay monitoring (unlike *ping*'s predefined path limitation).

We present SDProber—a software-defined prober for proactive delay measurements in SDN. SDProber uses probe agents to emit probes and measure the travel times of the probes. To avoid complex route computations, SDProber employes a novel method that combines *pseudo random walk* of the probes with binary exponential backoff. (In Section 3.2.2 we explain why a "pseudo" random walk is employed.) The probes traverse the network in a random walk over a weighted graph. The weights are adapted to route more probes via links whose lower limit is unsatisfied, and less probes via links whose the upper limit has been exceeded.

The goals of SDProber are as follows: *(1)* inspect links at specified rates, *(2)* reduce the total number of probe packets when monitoring the network, *(3)* minimize the number of excess packets through links, and *(4)* detect delays as early as possible. In our experiments, we show that in SDProber the expected number of probe packets going through a link is close to the observed number, which means that the underlying probabilistic model of the random walk is a correct representation of the actual probing. We compared SDProber to two baseline methods that send probe packets through shortest paths. SDProber uses 4–16 times fewer probe packets than the baseline methods and sends 10–62 times fewer surplus packets through links. Notwithstanding this significant cost reduction, SDProber is as fast as one baseline method and much faster than the other method, for detecting all the delays.

2 NETWORK AND DELAYS

A network is represented as a directed graph $G = (V, E)$ where the nodes are routers and switches. The edges are communication links between nodes. Given two nodes s_1 and s_2, $delay(s_1, s_2)$ is the time it takes for a packet sent from s_1 to arrive at s_2. When the measured delay for a link is higher than expected, in comparison to historical delays, we refer to the link as *delayed*.

In proactive monitoring, the links of the network are inspected periodically, e.g., using probe packets. The rate of inspection per each link, i.e., probe packets per link, should not exceed a given upper bound, to prevent a waste of resources, and it should not go below a certain lower bound,

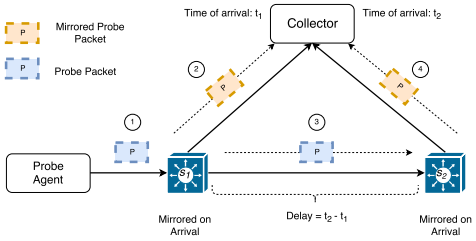

Figure 1: Schematic view of the mirroring process.

to prevent a case where delays are not detected for a very long time. The network operator specifies the minimum and maximum rates of probe-packet dispatching per link. This is specified as a *rate constraint* of the form (*min-rate*, *max-rate*).

Problem definition: The input is a network G with rate constraints on edges, and a cost constraint C that specifies the total probe packets per minute. The goal is to probe G such that probe rates would satisfy the rate constrains and the cost constraint C.

Sending probes via predefined paths could be problematic when paths consist of links that should be probed at different rates. Computing a set of paths that satisfies the probe-rate constraints is complex, expensive in terms of running times (essentially, NP-hard), and inflexible, in the sense that any change may require computing a different set. SDProber solves this by routing probe packets stochastically, based on the probe rates, in a random-walk fashion.

3 OVERVIEW OF SDPROBER

In this section, we present an overview of SDProber.

3.1 Delay Measurement

The delay between two given nodes s_1 and s_2 is measured by SDProber using probe packets (similar to [17]). A schematic representation of the process is depicted in Fig. 1. (1) A *probe agent* sends a probe on a path via s_1 and s_2. (2) When the probe packet arrives at s_1, it is mirrored, and the clone is sent to a *collector*. (3) The probe packet is forwarded to s_2. (4) Upon arrival at s_2, the probe packet is mirrored and the clone is sent to the collector. Let t_i be the time of arrival at the collector of the mirrored packet from s_i, for $i \in \{1, 2\}$. The estimated delay, is the time difference $t_2 - t_1$.

The measured time difference depends on the travel time of the mirrored packets from s_1 and s_2 to the collector. Measuring this one-way travel is hard, however the round trip from the collector to a node and back can easily be measured using ping. Let $t_1^{\leftrightarrow}$ and $t_2^{\leftrightarrow}$ be the round trip times between the nodes s_1, s_2 and the collector. Let $t_1^{\rightarrow}$ and $t_2^{\rightarrow}$ be the one-way trip times from the nodes s_1 and s_2 to the collector. Then, $t_2 - t_1 = delay(s_1, s_2) + t_2^{\rightarrow} - t_1^{\rightarrow}$. Clearly,

$t_1^{\leftrightarrow} \geq t_1^{\rightarrow}$ and $t_2^{\leftrightarrow} \geq t_2^{\rightarrow}$. So,

$$t_2 - t_1 \leq delay(s_1, s_2) + t_2^{\rightarrow} \leq delay(s_1, s_2) + t_2^{\leftrightarrow}$$

$$t_2 - t_1 \geq delay(s_1, s_2) - t_1^{\rightarrow} \geq delay(s_1, s_2) - t_1^{\leftrightarrow}$$

Accordingly,

$$delay(s_1, s_2) - t_1^{\leftrightarrow} \leq t_2 - t_1 \leq delay(s_1, s_2) + t_2^{\leftrightarrow}$$

That is,

$$t_2 - t_1 - t_2^{\leftrightarrow} \leq delay(s_1, s_2) \leq t_2 - t_1 + t_1^{\leftrightarrow}$$

This provides a bound on the error for a measured $delay(s_1, s_2)$. For example, if the measured values $t_2 - t_1$, $t_1^{\leftrightarrow}$ and $t_2^{\leftrightarrow}$ are 10 milliseconds, 2 milliseconds and 1 millisecond, in correspondence, then $delay(s_1, s_2)$ is at least 9 milliseconds and at most 12 milliseconds. See in [15] how to bound delay estimation errors in ISP networks.

3.2 System Architecture

SDProber sends probe packets repeatedly to measure delays in different parts of the network. The mirrored packets are collected at the collector, to compute the expected delay per link or path. This is used for creating a basis of comparison to detect anomalous delays. The architecture of the system is presented in Fig. 2. Next, we describe the different components of SDProber.

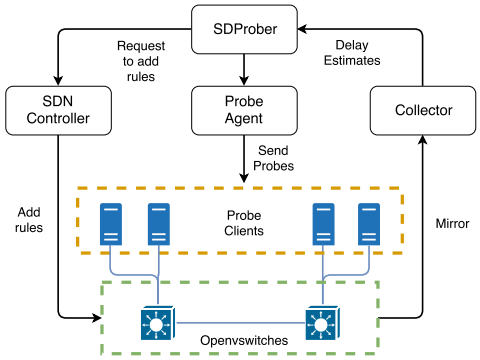

Figure 2: System architecture.

3.2.1 Probe Agent. The probe agent is responsible for crafting and dispatching probe packets, by activating probe clients. The number of probe clients can vary. Currently, a probe client is attached to every node. Alternatively, a small number of probe clients can be used. Each probe packet will be emitted from a probe client to the destined starting node s_1 and continue along the probing route. The TTL should be set appropriately.

The probe packets are marked to distinguish them from genuine traffic and to associate mirrored packets to their probing task. Each probe has a unique ID, in its payload.

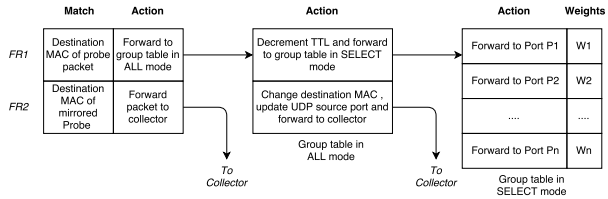

Figure 3: OpenFlow rules and group tables in SDProber.

The collector groups mirrored packets by this ID and distinguishes groups of mirrored packets of different probes from one another. This allows the collector to reconstruct the path for each probe packet. Differently from probe packets, mirrored packets should not be mirrored (on their way to the collector). In SDProber this is done by assigning a unique destination MAC addresses to probe packets and a unique destination MAC addresses to mirrored packets (the MAC-address field was chosen arbitrarily, and any available field can be used in lieu of it). To further restrict the traversal of the probe packets, the probe client sets the time to live (TTL) field in the packet header to a predefined limit.

3.2.2 SDN Controller and Open vSwitches. The underlying elements of the network are Open vSwitches (OVS) with an OpenFlow programming interface. Each OVS is identified by a *datapath ID* (DPID) known to the SDN controller. For adaptive measurements, SDProber routes probe packets in a random walk fashion. To do so, it uses a combination of group tables and match-action rules. OpenFlow's group tables are designed to execute one or more buckets for a single match, where each bucket consists of one or more actions. Group tables can be operated in different modes: ALL, SELECT, INDIRECT, and FAST FAILURE. SDProber uses group tables in ALL and SELECT modes. Group tables in ALL mode execute all the buckets, and group tables in SELECT mode execute a selected bucket. The selection is based on field values of the packet and weights that are given to the groups. Note that the selection is uniform when weights are equal.

Fig. 3 illustrates how OpenFlow forwarding rules and group tables handle probe packets. SDProber adds two match-action forwarding rules, *FR1* and *FR2*, to each OVS. The rules forward probe packets to group tables or to the collector. When a probe packet arrives at a switch, rule *FR1* forwards the probe packet to a group table in ALL mode, containing two buckets.

First, a mirrored packet is created, with a destination MAC address of a mirrored packet and a UDP source port equal to the DPID of that switch. The mirrored packet is routed to the collector. Second, the TTL of the probe packet is decremented and the probe is forwarded to the group table that operates in SELECT mode, where each bucket has a weight. For each

forwarded packet, the OVS chooses a bucket and executes the actions in the bucket. Each bucket contains a forwarding rule to a different neighbor node (a different port). The buckets are selected arbitrarily, per a hash of field values, in proportion to the weights.

OVS hashes several fields including MAC addresses, IP addresses, VLAN ID, Ethernet type, protocol (UDP in probe packets)—to choose a bucket in a group table. The hash values are cached by the OVS, to uniformly select buckets per each flow. Hence, to add randomness to the bucket selection, the probe agent assigns a unique source MAC address to each probe packet. Note, however, that in repeating visits of a probe packet at a node, the same actions are applied at each visit. Hence, the traversal is a *pseudo* random walk. A real random walk can be implemented by applying rules that take the TTL into account, but this would require adding more rules to each node (it will multiply the number of rules by the TTL limit), which is too expensive and unnecssary. When a mirrored probe packet arrives at a switch, match-action rule *FR2* forwards the probe packet to the collector. This prevents the unnecessary mirroring of already mirrored packets.

3.2.3 Collector. Mirrored probe packets reach the collector. The collector records the arrival time, extracts the UDP source from the header and gets the unique identifier from the payload. The mirrored packets are grouped by the identifier of the probe. If all the mirrored packets arrive at the collector, then the number of groups is equal to the total number of probe packets, and the number of packets in each group is equal to the initial TTL limit. After grouping, the collector computes the traversed path of each probe packet by ordering the mirrored packets of each group based on DPID values and the known network topology. The recorded times of arrival of the ordered mirrored packets are used for estimating the delay for each link on the path.

The collector stores the measured times and uses the stored data to compute the mean and the variance of the delay. By using reservoir sampling (see [18]), a random sample can be be used in lieu of the complete history, to reduce the storage space.

4 MONITORING BY RANDOM WALK

SDProber needs to satisfy the min-rate and max-rate constraints when routing probe packets. Computing a set of paths that satisfies all the constraints is computationally expensive and not always possible. Instead, in SDProber the probe packets perform a random walk (see [14]) over a weighted graphs.

In the random walk, the initial node and each traversal step are selected randomly, per probe. The link-selection probabilities are proportional to the weights of forwarding rules. The path length is limited by setting the TTL field to a particular value, say 10 steps. It determines the number of inspected links per probe packet, to reduce the rate at which probe packets are crafted and dispatched. Increasing the TTL also allows reducing the number of probe clients.

The initial node is selected randomly, possibly non-uniformly, as follows. Let n be the number of nodes in the network G, i.e., $|V| = n$, where each node $v_i \in V$ has a weight w_i. Let $W = \sum_{i=1}^{n} w_i$ be the sum of weights. The probability of selecting node v_i is w_i/W. To implement this, in each selection of a node, a number x in the range $[0, 1)$ is picked uniformly. The i such that $\frac{\sum_{j=1}^{i-1} w_j}{W} \leq x < \frac{\sum_{j=1}^{i} w_j}{W}$ is discovered, and v_i is the selected node.

When forwarding probe packets, the link (port) for the next step is chosen proportionally to the weights assigned to forwarding rules in the OpenFlow's SELECT-mode group tables. (See Section 3.2.)

To control the inspection rates, we need to estimate the number of probes passing through each link for a given number of emitted probes. This is done as follows. First, we compute visit probabilities for nodes. Let P_0 be a vector such that $P_0[i]$ is the probability of selecting v_i as the initial node, for $1 \leq i \leq n$. The *transition matrix* of G is an $n \times n$ matrix $M = (p_{ij})_{1 \leq i,j \leq n}$, where p_{ij} is the probability of forwarding the probe packet from v_i to v_j. Note that for each node v_i, the array $(p_{i1}, \ldots, p_{in})$ specifies the probabilities for the next step after reaching v_i. Thus, $p_{ij} = 0$ if v_i and v_j are not neighbors, and $\sum_{j=1}^{n} p_{ij} = 1$, for all $v_i \in V$.

Given the initial probabilities P_0 and the transition matrix M, $P_1 = (M^T)P_0$ is the vector of probabilities of reaching each node after one step of the random walk. By $P_t = (M^T)^t P_0$ we denote the probability of reaching each node after t steps of the random walk.

The probability of traversing a link (v_i, v_j) in a random walk of k steps is the probability of reaching node v_i in step t and proceeding to node v_j in step $t + 1$, for some $0 \leq t < k$. We denote this probability by $p\text{-}traverse_{ij}$. That is, $p\text{-}traverse_{ij} = \sum_{t=0}^{k-1} (P_t)_i (p_{ij})$, since $(P_t)_i$ is the probability of reaching node v_i at step t, and p_{ij} is the probability of forwarding to v_j a packet that arrived at v_i.

In the random walk approach, we do not need to conduct complex computations to craft probe packets or change them as they traverse the graph. If network changes require adjustments of probe rates, we merely have to alter the node weights of the initial node selection or the weights in the group tables.

5 WEIGHT ADAPTATION

In adaptive monitoring, weights that affect the random walk are adjusted to aid satisfying the rate constraints. SDProber modifies the weights iteratively using binary exponential

backoff. The iterations continue indefinitely, as long as the monitoring continues.

Link-weight adaptation. The weights are adapted to satisfy the probing constrains. Weights are doubled (halved) when the probing rate is below (above) the minimum (maximum) rate. Adding (reducing) weight increases (decreases) the probability of selecting the link in the next iteration.

Rates within the limits specified by the rate constraints are adjusted after each iteration. The weight of a link with a normal delay is decreased by half. The weight of each uninspected link is multiplied—by α if it was delayed during the last k iterations, and by 2, otherwise. The value k is configurable. By that, historically delayed links could receive a higher weight than links with no history of delays, to be visited more frequently.

Node-weight adaptation. Node weights should be modified to reflect the changes in links. The weight of a node with a link below the minimum rate is doubled, to increase the chances of visiting this node in the next iteration. Otherwise, the node weight is halved, to prevent traversal via links whose min-rate constraint is already satisfied. The weight is also halved for nodes with *(1)* a link whose probing rate is already greater than max-rate and *(2)* no links whose rate is less than min-rate.

6 BASELINE METHODS

Commonly, probe packets are sent via the shortest path between two nodes [6, 23]. Hence, we compare SDProber to two baseline methods in which each probe traverses the shortest path between selected nodes. We present now the baseline methods.

Random Pair Selection (RPS). In each iteration of RPS, pairs of source and destination nodes are selected randomly. The probe packets are sent via the shortest path from the source to the destination. The probe packets are mirrored to the collector at each hop, and the collector uses the arrival times of packets to estimate the delay on links, as in SDProber. In each iteration, the pair of source and destination nodes is selected uniformly from the set of pairs that have not been selected previously, till all the links are probed. This is repeated for the duration of the monitoring.

Greedy Path Selection works in a greedy fashion, iteratively. Initially an empty set *Visited* is created. In each iteration, for each pair of nodes, the *weight* of the shortest path P between these nodes is $\sum_{e \in P \text{ and } e \notin Visited} \textit{min-rate}(e)$, that is, the sum of the min rate values of all the unvisited links on the path. The path with the maximal weight is selected and its links are added to *Visited*. The probe packet is sent from the source to the destination of the selected path. The process ends when *Visited* contains all the links of the network. This is repeated while the monitoring continues.

7 EVALUATION

To test SDProber, we conducted a series of experiments. The goals are to show that *(1)* SDProber provides control over the link probing rates, *(2)* SDProber satisfies the min-rate constrains with a lower cost (in terms of the total number of packets) than the baseline methods, without increasing the delay deection time, and *(3)* given historical delay data, α can be tuned to improve performances.

Setting. The tests were executed on Mininet [4] with Open vSwitch 2.7.2 and a RYU controller [13]. We used a publicly-available real topology (the largest one in Topology-Zoo [8]), with 196 nodes and 243 links. A probing iteration was launched every 30 seconds. After every iteration, the weights of nodes and links were adjusted (see Section 3.2.2) and the rules were updated (see Section 5).

We tested three probing profiles, in which min-rate, max-rate and the the difference between them vary. The profiles are (*small*, 16, 20), (*medium*, 8, 28) and (*large*, 20, 60), where each tuple contains the name of the profile, min-rate and max-rate, respectively. Rates are in packet per minute (ppm). In each profile, the specified rates were assigned to all the links. In actual networks, the network operator can set probing rates based on SLA, the frequency of delays, etc.

In the presented results, the TTL limit was 10. In additional experiments, which are not presented here, we tested the effect of the TTL on the results and saw that as we increase the initial TTL, the detection time of delays decreases, but as the TTL approaches 10, the decrease gets smaller, and beyond 10 it is negligible. This is due to the hashing implementation by OVS, as discussed in Section 3.2.2.

7.1 Control over Inspection Rates

SDProber controls the number of probe packets through each link by adjusting the link weights. The probability of a packet to go through a link (*p-traverse*) is as described in Section 4. An important question is whether the probabilistic model is correlated with the actual traffic flow. To test this, we measured the number of probe packets going via each link, and compared the expected count with the measured one. In this test, the total number of emitted probes per iteration was equal to the sum of the min-rates over all the links. Each measured number of packets is the average of five executions and each execution is limited to ten iterations. Fig. 4 presents the results for the three probing profiles. The gray line depicts the expected values, which is the probability multiplied by the number of probe packets and the TTL. The graphs show that pseudo random walk provides inspection rates that are close to the expected values. For all the profiles, the error is within a range of around ±10% from the expected value. Thus, the probabilistic model provides a reliable description of the traffic flow. We conducted additional experiments,

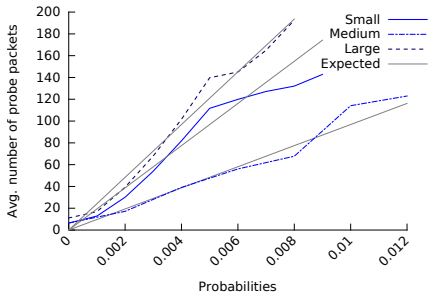

Figure 4: Actual probing rate versus expected rate, for links with different traversal probability.

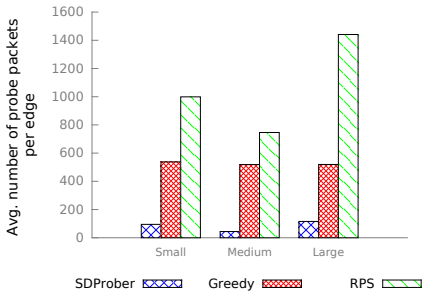

Figure 5: The average number probe packets per link when satisfying the min-rate constraints.

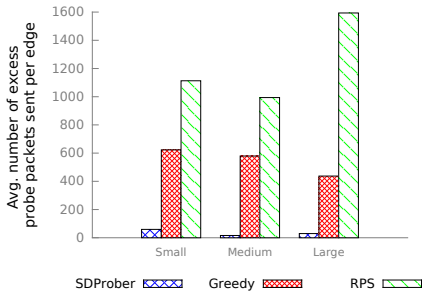

Figure 6: The number of surplus probe packets per link when satisfying the min-rate constraints.

with other probing rates, and the results were the same. The probabilities and their range depend on the topology and the rate constraints. Also, a small number of packets per iteration can lead to weight adjustments and an increase in the probabilities.

7.2 Cost Effectiveness

A main advantage of SDProber over the baseline methods is its ability to monitor the network with a low cost, in terms of the overall number of emitted probe packets. The next experiments show this. For each method, we increased the number of emitted probe packets per iteration till satisfying all the min-rate constraints of the links. Emitting fewer packets means that the constraints are satisfied with a lower cost. The results are presented in Fig. 5. SDProber sends fewer probe packets than Greedy and RPS, to satisfy the min-rate constrains. A reduction by a factor of 4.48–11.77 and of 10.52–12.44 is achieved in comparison to Greedy and RPS, respectively. The effectiveness of SDProber is achieved by adapting the weights to avoid probing links whose min-rates were already satisfied.

SDProber satisfies max-rate constraints more strictly than Gready and RPS. To show this, Fig. 6 depicts the average number of excess probe packets per link, i.e., packets that cause the probing rate to exceed max-rate. SDProber sends 10.46–36.31 and 18.69–62.29 times fewer surplus packets than Greedy and RPS, respectively.

7.3 Adjusting α

The parameter α provides control over the weight adaptation, to inspect frequently-delayed links (FDL) at a higher rate than other links. We tested the effect of α on a network with 10% delays, where f% are FDL, i.e., delays repeat for these links in different iterations. We varied the α values from 2 to 4 with steps of 0.4 for the *small* probing profile and show its influence on the detection time for $f \in \{0\%, 50\%, 100\%\}$ in Fig. 7. The number of packets per iteration was small (200), to emphasize the effect of frequent probing of FDL. Each reported time is the average of five executions and each execution is limited to ten iterations. When there are FDL, i.e., $f \in \{50\%, 100\%\}$, increasing α reduces the detection time, by 2.84% and 5.58% on an average. But the reduction in detection time decreases with the increase in *alpha* because other parameters (number of probe packets, f and rate constraints) influence the detection time as well. However, when there are no FDL, i.e., $f \in \{0\%\}$, decreasing α increases the detection time by 3.91% on an average and $\alpha = 2$ yields the fastest detection time.

7.4 Detection Time

To test how fast delays are detected, we evaluated the time it takes to detect all the delayed links. The percentage of delayed links in different runs was varied from 10% to 100%, at increments of 10% (in all cases, the entire network should be inspected to detect all delays). In each experiment, the number of packets available per iteration was equal to the sum of min-rate values over all the links—this ensures that there are enough packets per iteration to probe all the links at min-rate. Fig. 8 presents the results for the *medium* probing profile. The results are similar for the other two probing profiles. Each reported time is the average of five executions and iterations are continued till all the delayed links have been detected. RPS detects delays slightly slower than SDProber, but requires many more packets to satisfy the min-rate constraints. RPS and SDProber detect delays about twice faster than Greedy. SDProber conducted less than 3 complete iterations (weight updates) during this test.

The reason for the slow detection by Greedy is that links with a low weight are visited last, and a delay on such a link is detected behindhand. In SDProber, the weight adjustment guarantees frequent probing of all the links, according to the min-rates. Note that RPS ignores the weights, so regardless of where the inspection rates are violated, the entire network is

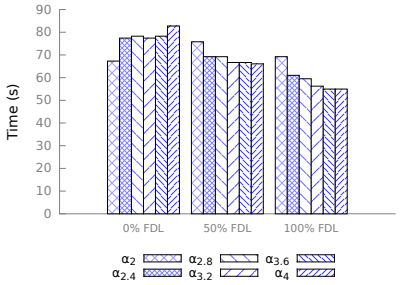

Figure 7: Detection time of delayed links per α.

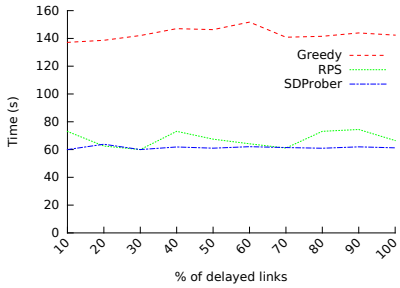

Figure 8: Detection time of delayed links.

inspected, including links which have already been inspected. So, detecting all the delays with RPS is relatively fast, but it has a high cost.

8 WEIGHT ALLOCATION

SDProber can be configured to use different weight-allocation strategies, e.g, *static* or *adaptive*—we investigated the adaptive case. Previous studies [5, 11] have shown that link failures can be learned. Accordingly, a failure probability can be associate with each link, to determine static weights for the random walk (future work). However, in static weight allocation, links with a low failure probability would be inspected at a very low rate—this reduces costs but may lead to late detection of failures in links with a low weight.

As an example use case of dynamic weight allocation, consider AT&T FlexWare—a *Network Function on Demand* architecture. A customer could deploy dozens or hundreds of virtual network functions (VNFs) on FlexWare devices, and the network between them would be monitored by SDProber. Based on full knowledge of the VNFs, the connections between VNFs will be monitored at rates that are specified by the customer, to comply with the SLA. When a new service is deployed, it can be accommodated by weight adaption, easily and promptly. A similar approach can be applied to a hybrid environment where only some of the nodes are OVS.

9 RELATED WORK

Network measurements, including delay measurement, received a lot of attention over the years [2, 9, 12]. However, these papers do not show how to use SDN capabilities or centralized control for effective proactive measurement of the delay. Recently, with the advent of SDN, there is a growing interest in measurements that are adapted to SDN [1, 3, 10, 16, 19, 20, 22].

Several systems utilize mirroring for measurements. NetSight [7] uses mirroring to gather information about the trajectories of all the packets in a network. However, their method is only suitable for small networks, and does not scale. Everflow [24] provides a scalable sampling of packets in datacenter networks. They, however, require specific hardware, e.g., multiplexers, to support their sampling. Furthermore, their method is reactive, to be used in a response to an event, while SDProber is proactive, to monitor the network in an economic way and detect high delays early.

Using probe packets to measure latencies in OpenFlow-based networks was studied in [17, 21]. SLAM [21] uses the time of arrival of OpenFlow packetin messages at the controller to estimate the delay between links. However, this approach requires knowledge of traffic patterns, e.g., to distinguish between lack of packetin messages and delays that obstruct packetin messages, so it may not be applicable to networks which are not a datacenter. The OpenNetMon [17] system provides per-flow metrics, like throughput, delay and packet loss, for OpenFlow networks. It uses probe packets and a collector for delay measurement. Pingmesh [6] uses ping to measure delays in a datacenter. Unlike SDProber, OpenNetMon and Pingmesh do not provide an adaptive measurement framework.

10 CONCLUSION

We presented SDProber—a prober for proactive measurement of delays in SDN. SDProber provides control over the inspection rates of links, by routing probe packets according to min-rate and max-rate constraints, while utilizing each probe packet to inspect several links. To satisfy the constraints without expensive computations, the dispatched probes conduct a random walk over a weighted graph. The weights are modified using binary exponential backoff to achieve the desired probing rate for each link. Furthermore, weight adaptation can use the history of delays, to inspect frequently-delayed links at a higher rate than other links.

We demonstrate how random walk is implemented using OpenFlow. In our evaluation, we show that SDProber is more effective than baseline methods by sending significantly fewer probe packets without increasing the time it takes to detect delays and while reducing the number of excess packets through links.

REFERENCES

[1] Alon Atary and Anat Bremler-Barr. 2016. Efficient Round-Trip Time Monitoring in OpenFlow Networks. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 1–9.

[2] Jean-Chrysotome Bolot. 1993. End-to-end packet delay and loss behavior in the Internet. *ACM SIGCOMM Computer Communication Review* 23, 4 (1993), 289–298.

[3] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. 2011. DevoFlow: Scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review* 41, 4 (2011), 254–265.

[4] Rogério Leão Santos De Oliveira, Ailton Akira Shinoda, Christiane Marie Schweitzer, and Ligia Rodrigues Prete. 2014. Using mininet for emulation and prototyping software-defined networks. In *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*. IEEE, 1–6.

[5] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. *SIGCOMM Comput. Commun. Rev.* 41, 4 (Aug. 2011), 350–361. https://doi.org/10.1145/2043164.2018477

[6] Chuanxiong Guo, Lihua Yuan, Dong Xiang, Yingnong Dang, Ray Huang, Dave Maltz, Zhaoyi Liu, Vin Wang, Bin Pang, Hua Chen, et al. 2015. Pingmesh: A large-scale system for data center network latency measurement and analysis. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 139–152.

[7] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks.. In *NSDI*, Vol. 14. 71–85.

[8] Simon Knight, Hung X Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. 2011. The internet topology zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775.

[9] Kevin Lai and Mary Baker. 2000. Measuring link bandwidths using a deterministic model of packet delay. *ACM SIGCOMM Computer Communication Review* 30, 4 (2000), 283–294.

[10] Jieyu Lin, Rajsimman Ravichandiran, Hadi Bannazadeh, and Alberto Leon-Garcia. 2015. Monitoring and measurement in software-defined infrastructure. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 742–745.

[11] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, Yashar Ganjali, and Christophe Diot. 2008. Characterization of Failures in an Operational IP Backbone Network. *IEEE/ACM Trans. Netw.* 16, 4 (Aug. 2008), 749–762. https://doi.org/10.1109/TNET.2007.902727

[12] Sue B Moon, Paul Skelly, and Don Towsley. 1999. Estimation and removal of clock skew from network delay measurements. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 1. IEEE, 227–234.

[13] SDN Ryu. 2016. Framework. https://osrg.github.io/ryu/. (2016).

[14] Frank Spitzer. 2013. *Principles of random walk*. Vol. 34. Springer Science & Business Media.

[15] Olivier Tilmans, Tobias Bühler, Stefano Vissicchio, and Laurent Vanbever. 2016. Mille-Feuille: Putting ISP traffic under the scalpel. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 113–119.

[16] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. 2010. OpenTM: traffic matrix estimator for OpenFlow networks. In *International Conference on Passive and Active Network Measurement*. Springer, 201–210.

[17] Niels LM Van Adrichem, Christian Doerr, and Fernando A Kuipers. 2014. Opennetmon: Network monitoring in openflow software-defined networks. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 1–8.

[18] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57.

[19] An Wang, Yang Guo, Fang Hao, T. V. Lakshman, and Songqing Chen. 2015. UMON: Flexible and Fine Grained Traffic Monitoring in Open vSwitch. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT '15)*. ACM, New York, NY, USA, Article 15, 7 pages. https://doi.org/10.1145/2716281.2836100

[20] Philip Wette and Holger Karl. 2013. Which Flows Are Hiding Behind My Wildcard Rule?: Adding Packet Sampling to Openflow. *SIGCOMM Comput. Commun. Rev.* 43, 4 (Aug. 2013), 541–542. https://doi.org/10.1145/2534169.2491710

[21] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V Madhyastha. 2015. Software-defined latency monitoring in data center networks. In *International Conference on Passive and Active Network Measurement*. Springer, 360–372.

[22] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V Madhyastha. 2013. Flowsense: Monitoring network utilization with zero measurement cost. In *International Conference on Passive and Active Network Measurement*. Springer, 31–41.

[23] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Automatic Test Packet Generation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*. ACM, 241–252.

[24] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y. Zhao, and Haitao Zheng. 2015. Packet-Level Telemetry in Large Datacenter Networks. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 479–491. https://doi.org/10.1145/2829988.2787483