SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks

Yilong Geng, Shiyu Liu, and Zi Yin, *Stanford University;* Ashish Naik, *Google Inc.;*
Balaji Prabhakar and Mendel Rosenblum, *Stanford University;* Amin Vahdat, *Google Inc.*

https://www.usenix.org/conference/nsdi19/presentation/geng

This paper is included in the Proceedings of the
16th USENIX Symposium on Networked Systems
Design and Implementation (NSDI '19).

February 26–28, 2019 • Boston, MA, USA

ISBN 978-1-931971-49-2

Open access to the Proceedings of the
16th USENIX Symposium on Networked Systems
Design and Implementation (NSDI '19)
is sponsored by

SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks

Yilong Geng[1], Shiyu Liu[1], Zi Yin[1], Ashish Naik[2],
Balaji Prabhakar[1], Mendel Rosenblum[1], and Amin Vahdat[2]

[1]Stanford University
[2]Google Inc.

Abstract

It is important to perform measurement and monitoring in order to understand network performance and debug problems encountered by distributed applications. Despite many products and much research on these topics, in the context of data centers, performing accurate measurement at scale in near real-time has remained elusive. There are two main approaches to network telemetry–switch-based and end-host-based–each with its own advantages and drawbacks.

In this paper, we attempt to push the boundary of edge-based measurement by scalably and accurately *reconstructing* the full queueing dynamics in the network with data gathered entirely at the transmit and receive network interface cards (NICs). We begin with a Signal Processing framework for quantifying a key trade-off: reconstruction accuracy versus the amount of data gathered. Based on this, we propose SIMON, an accurate and scalable measurement system for data centers that reconstructs key network state variables like packet queuing times at switches, link utilizations, and queue and link compositions at the flow-level. We use two ideas to speed up SIMON: (i) the hierarchical nature of data center topologies, and (ii) the function approximation capability of multi-layered neural networks. The former gives a speedup of 1,000x while the latter implemented on GPUs gives a speedup of 5,000x to 10,000x, enabling SIMON to run in real-time. We deployed SIMON in three testbeds with different link speeds, layers of switching and number of servers. Evaluations with NetFPGAs and a cross-validation technique show that SIMON reconstructs queue-lengths to within 3-5 KBs and link utilizations to less than 1% of actual. The accuracy and speed of SIMON enables sensitive A/B tests, which greatly aids the real-time development of algorithms, protocols, network software and applications.

1 Introduction

Background and motivation. Measurement and telemetry are long-standing important problems in Networking; there's a lot of research on these topics and there are several products providing these functionalities (e.g., [42, 21, 19, 56, 55, 57, 27, 13, 20, 50, 44, 40, 49, 7, 1, 2, 3, 4]). The primary use cases are monitoring the health of networks, measuring their performance, billing, traffic engineering, capacity planning, troubleshooting in the case of breakdowns or failures, and for detecting anomalies and security threats. The key challenges are: (i) accuracy: how to accurately observe and measure events or phenomena of interest; (ii) scalability: how to scale the measurement method to large networks, involving hundreds or thousands of nodes and high line rates, hence a very large "event frequency"; and (iii) speed: how to perform accurate and scalable measurement in near real-time as opposed to offline. Since these are conflicting requirements, most solutions seek to make effective trade-offs.

Measurement methods can be classified as "switch-based" or "edge-based". Switch-based methods can be approximate or exact. We survey the literature on this topic in Section 7. For now, it suffices to say that most early work (and products; e.g., NetFlow [50] and sFlow [44]) consider approximate measurement since accurate measurement was deemed prohibitively expensive. These methods only give approximate counts of packets/bytes passing through a single switch, requiring a lot of extra processing to stitch together network-wide, flow-level views. Further, they also require extra bandwidth to move the measurement data to the network's edge for processing. Recent developments in programmable switches and in-band network telemetry [58, 30, 32, 28] enable accurate, per-packet measurement. However, they generate a lot of data (per-packet, per-switch), whereas we shall see that network phenomena of interest can be captured with a lot smaller data. The effectiveness of INT also relies on all nodes being able to perform it. Finally, because switches are not adjacent to the end-hosts (in the way that NICs are), they cannot easily relate network bottlenecks to application performance.

Edge-based methods record "events" at end-hosts with little or no help from the network. The events are used to infer some network state that is of interest to an application or to the operator. Since storage is distributed and resources

in the end-hosts are abundant, these methods are inherently scalable. The question is how much network state can be inferred and how accurately? Existing work, surveyed in Section 7, only obtains a partial or approximate view of the network state from edge observations, such as end-to-end delay distributions, which link dropped a packet, detecting traffic spikes, silent packet drops, load imbalance, or routing loops.

By contrast, our work, which is also edge-based, obtains a near exact reconstruction of network state variables.[1] That is, we obtain key variables like queuing delays, link utilizations and queue/link compositions over small time intervals and on a per-packet or per-flow basis. Our approach is based on *network tomography*.

Tomography. The goal of network tomography is to use the "individual records"[2] of unicast or multicast probes and packets collected at the *edge* of the network and determine *internal* network state quantities such as delays and backlogs at individual links. The general philosophy of network tomography is this: While each record conveys a limited amount of information about network conditions, it may be possible to combine the records of all the probes or packets to get a detailed picture of internal network state and conditions.

Despite much research (surveyed in Section 7), network tomography hasn't proved successful in wide area networks. As [25] notes, one major impediment is ignorance of the underlying network topology. This leads tomography algorithms to make unrealistically simple assumptions, which, in turn, lead to inaccurate inferences. Even if the exact topology were known, the end-to-end traversal times in the wide area setting are at least a few milliseconds and typically a few tens of milliseconds, much longer than the queuing times at routers. So two probes whose network dwell times overlap might encounter quite different queuing times at a common router buffer on their path. Since the accurate determination of queuing times is infeasible in the wide area setting, [25] advocates determining their *distributions* instead.

Reconstructing data center state variables. We revisit network tomography in the data center context. By restricting ourselves to data centers, we sidestep the problems plaguing tomography in wide area networks and obtain the following advantages. (a) A data center is operated by a single administrative entity, hence, the network topology is easy to know. The path followed by a probe or packet is also knowable (e.g., using traceroute or because the hash functions which assign packets to paths are known). (b) A modular network topology of the Clos type provides multiple paths between any pair of nodes, making congestion and bottlenecks sparse. (c) As a consequence of (a) and (b), the network traversal time of a probe or packet is dominated by queueing times at one or two queues and *the wire times are negligible*.[3]

It is important to note that we do not reconstruct the *instantaneous values* of network state variables, rather we reconstruct a *I-average* of these quantities, where I is a short interval (e.g., 0.25 msec–1 msec in 10–40 Gbps networks). In Section 2.1, we demonstrate that packet queuing times and backlog processes viewed at the granularity of packet enqueuing and dequeuing times are very noisy. By analyzing the queuing process in the frequency domain (specifically, by looking at its *power spectral density*), we propose to reconstruct the I-averaged queuing times and link utilizations and show that these quantities retain 97.5% of the power of the corresponding instantaneous quantities, and are practically the same in value except for the noise. A major benefit is that the I-averaged network state quantities are obtained with much less data and processing effort! For example, in a 10 Gbps, 256-server network with 3 tiers of switching operating at 40% load, going from per-packet to per-millisecond data capture at the edge reduces total storage by 60x and speeds up computation by 40x with negligible reduction in accuracy (see Table 1).

If it works in the data center setting, the advantages of a tomography-based measurement system are several: (i) it doesn't require any modification to the existing switching infrastructure since it only needs data to be gathered at the edge, and most current-generation NICs are able to timestamp packets at wirespeed, (ii) by injecting extra probes at roughly 0.3% of the link bandwidth to obtain network state information, its bandwidth overhead is negligible when compared with switch-centric approaches which need to send data collected at the switches to the network's edge, (iii) being edge-based, it is readily able to relate application-level performance to network bottlenecks, and (iv) most importantly, it has the potential to be accurate, scalable and near real-time.

Our contributions.

We propose SIMON, a sensing, inference and measurement system *for data centers* that reconstructs key network state variables such as queuing times, link utilizations and queue and link compositions (i.e., breaking down the packets in a queue according to flow ids). SIMON uses a mesh of probes to cover all the linearly independent paths in the network, and the delays of the probes in a reconstruction interval are processed by the LASSO inference algorithm [53] to obtain the queue size and other related variables. We present:

(1) A signal processing framework for analyzing the basic elements of tomography-based measurement methods (Section 2.1). The main finding is that queue sizes and wait times fluctuate noisily when viewed at packet enqueuing and dequeuing times, but a low-pass filtered version of these processes is both easier to reconstruct and carries more than

[1] "near exact reconstruction" is defined in Section 2.1.

[2] An individual record, described formally later, consists of the standard 5-tuple, transmit and receive timestamps, and the byte-count.

[3] For example, the propagation time is 5ns for 1 meter or 0.5 microsecs for 100 meters which is comparable to the raw switching (zero queuing) time at a node.

97.5% of the signal power.
(2) SIMON, a probe-based sensing, inference and measurement system for accurately determining network state variables (Section 3). In an ns-3 simulation of a 10 Gbps, 256 server DCN, SIMON achieves an RMSE of 4.14KB in reconstructing queue compositions, and a 1% error in reconstructing link utilization for each traffic class (see Figure 11). In a real-world 1 Gbps, 128 server testbed, SIMON achieves an RMSE of 5.1KB (just over 3 1500B packets) with respect to ground truth.
(3) Exploiting the hierarchical (Clos or fat-tree type) structure of modern data center topologies to devise a modular, fast version of SIMON (Section 4.1). The resulting speedup is 1,000x.
(4) Using the function approximation capability of multi-layered neural networks (NNs) [29] to hardware-accelerate SIMON, more specifically, to hardware-accelerate the LASSO inference algorithm used by SIMON. We find that even unoptimized NNs running on standard GPUs, give up to 5,000–10,000x acceleration (Section 4), enabling SIMON to run in near real-time.
(5) A verification of the accuracy and scalability of SIMON on a 128 server, 1 Gbps data center and a 240 server, 40 Gbps testbed. We comment on the deployment experience from these testbeds as well as a mixed 10G-40G, 288 server network (Section 6).

As a consequence of the speed and accuracy of SIMON, it can support sensitive A/B tests in near real-time, hence enabling the rapid development of algorithms, protocols, data center software and high-level applications. It can also be coupled with edge-based control in "Smart NICs" which are a recent and major focus of the industry [37, 10, 14].

2 Network reconstruction using tomography

In this section we describe how to use data sensed from the edge of the network (servers) to reconstruct key performance measures in a data center network (DCN). Figure 1 shows a 3-stage Clos (or fat-tree) DCN, with the path taken by a probe or data packet shown in red. We introduce the following concepts and terminology.
Probe. A probe is a 64 byte UDP or TCP packet sent from one server to another in the DCN. It travels on the same priorities and through the same queues as regular data packets. Its purpose is to incur the same queuing times as the payload in its priority and, hence, provide data for reconstruction.
Probe mesh. This is a graph connecting the N servers in a DCN to one another along whose edges probes are exchanged. Each server picks K other servers randomly and uniformly (without replacement) from the set of all servers. Suppose server i picked servers $j_1, ..., j_K$. Then i sends probes to each j_l ($1 \leq l \leq K$) at a frequency F Hz. Each j_l sends probes back to i at the same frequency. The path in the DCN followed by probes between any pair of servers is chosen uniformly at random from the set of all available paths between them, and independently of the choice of paths for other pairs.[4] The paths are chosen once and held fixed. Note that each server sends a total of $2KF$ probes per second on average.
Remark. We shall later demonstrate (Section 3) that with $10 \leq K \leq 20$ we can sample all the links (hence queues) in *any Clos (fat-tree) DCN*; that is, a constant value of K suffices. We shall also comment on F.
Handling Ethernet priorities. There are 8 priorities in Ethernet, the dominant L2 technology in DCNs. For convenience, in this paper we assume that the DCN uses only one priority, although the verification in the 40 Gbps testbed was conducted in a multi-priority setting. In order to handle multiple priorities, we simply launch a probe mesh in each priority and perform reconstruction per priority. The probes will use the same transport protocol (TCP or UDP) as the traffic in the given priority and encounter the same queuing delays as the payload. Indeed, the 40 Gbps testbed also employs priority flow control (IEEE 802.1Qbb), where packets in a priority may be paused. Reconstruction works in this case as well.
Individual record. The individual record of a probe or a data packet is captured at the transmitter network interface card (NIC) and the receiver NIC. It consists of the 5-tuple header information (source and destination addresses, source and destination port numbers, protocol port number), the transmit and receive timestamps at the corresponding NICs, and the length of the packets.
Remark. We assume that the clocks of all the NICs are accurately synchronized using techniques in [24], [31] or [33]. Note that, even though these techniques can synchronize clocks up to a 10s of nanoseconds, it suffices for our purposes that the clocks be synchronized to about 1 microsecond.

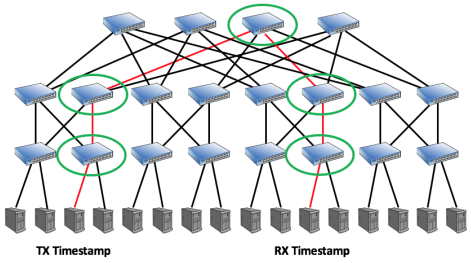
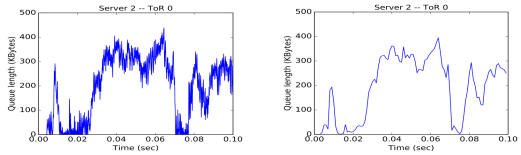


Figure 1: Reconstruction from Edge Timestamps

Output-queue assumption. Switches in DCNs can have queues on both the ingress and egress line cards and packets can queue at both places. However, switch implementations

[4] The paths of the probe mesh do not have to be chosen at random; indeed, it may sometimes be desirable to choose the path deliberately so as to cover the DCN's links more evenly or unevenly, depending on some objective. Further, probes from i to j can follow a different path in the DCN than probes from j to i. Whatever the choice, the paths of the probe mesh must cover all the links in the network uniformly, and cover all the "linearly independent paths" with an adequate number of probes passing through each link per unit time so as to enable a good reconstruction of that link. Random path selection is adequate for these purposes.

Figure 2: The network in ns-3 simulation setup (NS)



(a) Queue sampled every $1\mu s$ (b) The $1ms$ average

Figure 3: Queue length in a 10Gbps network

employ a fabric speedup [16], ensuring that most queuing takes place at the egress line cards. Throughout this paper we assume that queuing takes place in the output queues of switches, and measurements done in a real testbed in Section 6.2 validate this assumption. Hence, each directed link in the DCN has a queue associated with it; namely, the output queue that drains into it. [5]

2.1 What to reconstruct

Suppose we have a DCN with a total of N^q queues. Under some traffic load, let $Q_i(t)$ be the queue length at time t and $W_i(t)$ be the waiting (or queuing) time of a packet arriving at time t to the i^{th} queue. With a single priority $W_i(t) = Q_i(t)/L$, where L is the line rate in Bytes/sec, so the waiting time and queue size are related by a constant.[6] Consider a preliminary statement of the reconstruction problem:

Given the individual records of all the probe packets during some time $[0,T]$, obtain a reconstruction $\hat{Q}_i(t)$ of $Q_i(t)$ so as to minimize $\mathbb{E}\left[Q_i(t) - \hat{Q}_i(t)\right]^2$ for $t \in [0,T]$.

ns-3 simulation (NS) setup. To gain an understanding of what is involved in solving this problem, let us consider an ns-3 simulation of a 10 Gpbs, 3-layer, 256-server network. We shall reuse this simulation set up in the next section as well; hence, we shall refer to it as the NS setup. This network has 32 ToR switches (8 servers per rack), 32 spine layer 1 switches and 8 spine layer 2 switches, connected as shown in Figure 2. All switches are output-queued and have 1MB of buffering per queue. The DCN operates under an incast traffic[7] load of 40%. We have also used a "long flow" workload at 40% load, where one file is sent from a server to another randomly chosen one, and the file sizes are uniformly in [10MB, 1GB] with an average of 505MB.

$Q_i(t)$, measured every microsecond, is plotted in Figure 3a. (Note that packet enqueuing/dequeuing times on a 10G link are close to 1 microsecond.) The 1 ms-averaged version of $Q_i(t)$, $\bar{Q}_i(t)$, in Figure 3b) is obtained by averaging the 1000 consecutive values of $Q_i(t)$ in each millisecond. The two graphs are essentially identical; the $Q_i(t)$ is a noisy version of $\bar{Q}_i(t)$. We shall next show that there is a straightforward relationship between $Q_i(t)$ and $\bar{Q}_i(t)$; indeed, $\bar{Q}_i(t)$ has almost all of the *signal power* in $Q_i(t)$. In Section 3 we present a method for reconstructing $\bar{Q}_i(t)$ using the LASSO algorithm.

Averaging: empirical evidence. Define the autocorrelation function of $Q_i(t)$ as follows:

$$R_Q(\tau) = \mathbb{E}\left[Q_i(t+\tau)Q_i(t)\right], \text{ for } \tau \geq 0. \quad (1)$$

The autocorrelation function captures the rate of decay of correlations in $Q_i(t)$. The power spectral density (PSD) of $Q_i(t)$ equals[8]

$$S_Q(f) = \sum_{\tau=-\infty}^{\infty} R_Q(\tau) e^{-i2\pi\tau f}, \text{ for } |f\tau| < 1/2, \text{ or } |f| < 0.5 \text{ MHz}. \quad (2)$$

The PSD shows the amount of power in $Q_i(t)$ at different frequencies and is symmetric for positive and negative values of f in the range over which it is defined (-0.5 MHz $< f <$ 0.5 MHz in this case).

Plotting the PSD as a function of the frequency in Hz for $0 < f \leq 0.5$ MHz, we get the yellow curve in Figure 4a. By computing the PSD similarly for $\bar{Q}_i(t)$ and plotting it, we get the blue curve in Figure 4a. The blue curve exactly coincides with the yellow for -500 Hz $< f <$ 500 Hz, which means that the power at those frequencies in both $Q_i(t)$ and $\bar{Q}_i(t)$ are *identical*. Further, $\bar{Q}_i(t)$ has zero power in frequencies higher than 500 Hz; that is, averaging only removes the high frequency "noise" and preserves most of the power, 97.5% to be precise, of the microsecond-level signal $Q_i(t)$.

Remark. It is worth noting that the percentage of preserved signal power depends on the strength of the signal itself. The 97.5% number is obtained at 40% load with 1 MB switch buffers. Figure 4b shows the PSD measured with different network load and switch buffer size combinations. As can be seen, although the amount of power preserved after averaging (the low frequency signal power) varies, the power of the removed high frequency component remains constant: at $(12KB)^2$ over all frequencies higher than 500 Hz.[9]

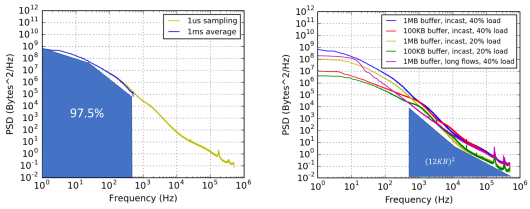
[5] Note that the only directed links with which no queues need to be associated are those connecting a NIC to a top-of-the-rack (ToR) switch. These links don't get oversubscribed.

[6] Note that the inference equations will be in terms of $W_i(t)$, as in equation (3). The queue wait times in each priority come from the probes in that priority. We're looking at the queue lengths here rather than queue wait times because the lengths vary directly with packet arrivals and departures.

[7] The incast traffic pattern is taken from DCTCP [6]. Each server maintains a certain level of load by requesting data simultaneously from a random number (30% 1, 50% 2 and 20% 4) of other servers, referred as the "fanout" of the request. The gap between adjacent requests are independent exponentials with load-dependent rate. The file sizes are distributed according to a heavy-tailed distribution in the range [10KB, 30MB] with an average file size of 2.4MB. We have used incast workload in this paper because it captures some generic scenarios (long file transfers and short RPC-type traffic) and causes congestion. In deployments we've used map-reduce-type batch workloads and RDMA-type traffic.

[8] Note that τ represents a 1 us interval; hence $1/\tau$ is 1 MHz.

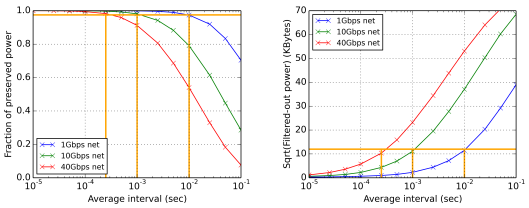
[9] At the moment we do not have an explanation about this empirical ob-



(a) In a network with 1MB buffer, 40% incast load

(b) In different networks and with incast and long flows workload

Figure 4: PSD of queue depths in 10Gbps networks



(a) The fraction of queue size power preserved

(b) The "noise" power or the power "filtered-out"

Figure 6: Power of queue size and noise processes at different reconstruction intervals

As empirical evidence from a real deployment, Figure 14 compares the PSD of the queue size process measured using NetFPGA in a 1 Gbps testbed with shallow-buffered switches. As discussed below, the averaging interval is a function of only the line rate and for a 1 Gbps line it is 10 ms. In Figure 14 we again see that at 40% load the PSD of the averaged queue process coincides with the PSD of the queue size process. Again the power of the portion that is filtered out, the "noise", is $(12KB)^2$.

Averaging: signal processing explanation. $\bar{Q}_i(t)$ is obtained precisely by passing $Q_i(t)$ through a low-pass filter with the pass band equal to $[-500$ Hz, 500 Hz$]$ [41]. This is easy to understand from Figure 5: $\bar{Q}_i(t)$ is obtained by (i) computing a running average of $Q_i(t)$ over a 1 ms window, and then (ii) down sampling the running average to 1 sample per ms. The frequency domain transfer functions ($H(f)$ and $X(f)$) corresponding to (i) and (ii) are shown in the same figure. Essentially, (i) and (ii) precisely amount to removing the frequency components in $Q_i(t)$ over 500 Hz and preserving the rest.

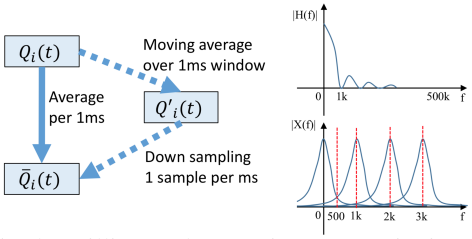


Figure 5: The millisecond-averaging process in time and frequency domains

Upshot. As a result of the above discussion, we shall reconstruct the averaged queue process $\bar{Q}_i(t)$ for each i rather than the packet-time-sampled queue process $Q_i(t)$. It remains to specify the "reconstruction interval"; we proceed to do this next.

Determining the reconstruction interval. The proper reconstruction interval for a given DCN turns to depend only on the *highest line rate* in the network, even when there are mixed link speeds. Intuitively, the faster the link speeds, the quicker the queues vary (or the higher the noise frequency), and the smaller is the reconstruction interval.

Figure 6a shows the percentage of power preserved by using different reconstruction intervals in networks with different link speeds at 40% load. As shown, to preserve more than 97.5% of the power, the reconstruction interval is 10 ms for 1Gbps links, 1 ms for 10 Gbps links and 250 μs for 40Gbps links. In other words, the reconstruction interval is inversely proportional to the maximum link speed of the network. Figure 6b shows the square root of the power filtered-out at different reconstruction intervals. Remarkably, as seen in the figure, this value is a constant at 12KB *at all link speeds* when the reconstruction interval equals 10 ms, 1 ms and 0.25 ms for 1G, 10G and 40G links, respectively. In other words, the "noise" power is $(12KB)^2$.

In summary, we shall aim to reconstruct the average queuing times for the all the queues in the switches for each reconstruction interval, which is determined by the maximum link speed of the network. The reconstructed average queues removes the high frequency noise and preserves most of the relevant information in the original queue signal.

3 The reconstruction algorithm

We now describe SIMON; specifically, we describe
(1) a system that uses the individual records of probes to reconstruct the averaged queue or wait time processes using LASSO, and
(2) uses the above and the individual records of data packets to determine link utilizations as well as queue and link compositions.

Preliminaries. Given a DCN, the first steps are to set up a probe mesh and to choose a reconstruction interval, I. Consider all the probes sent by all the servers in the reconstruction interval I (recall that we assume the clocks are all accurately synchronized). Let t_i^p, $i = 1,...,N^p$ be the ordered sequence of transmit timestamps of all the probes (regardless of source) sent in interval I, where N^p is the total number of probes transmitted in I. Let r_i^p be the receive timestamp of the probe transmitted at t_i^p. Note that some r_i^p may fall out-

servation that the removed noise power equals $(12KB)^2$. In the rest of the paper we use this $(12KB)^2$ to define the acceptable signal power loss caused by averaging and determine the averaging interval so as to achieve no more than this loss.

side I; this is fine.[10] Set $D_i^p = r_i^p - t_i^p$ to be the one way delay of probe i. Let t_j^d, r_j^d, D_j^d and N^d be the corresponding quantities for data packets transmitted in interval I. Denote by D^p and D^d the $N^p \times 1$ and $N^d \times 1$ the vectors of one way delays of all the probe and data packets transmitted in interval I, respectively.

By the output-queue assumption, preceding each link in the network we imagine there is a queue in each direction of the link. Recall that there are a total of N^q queues (counting both directions at each link). Let $q_k(I)$ denote the average queue size process at queue numbered i in interval I. Since we've fixed I, we abbreviate $q_k(I)$ to q_k. Similarly let w_k be the waiting time of a packet in queue k in interval I (recall $w_k = q_k/L$, where L is the line rate in Bytes/sec). Let Q and W denote the $N^q \times 1$ vectors of queue sizes and wait times in interval I, respectively.

Recall that we have assumed that the path taken by a probe or a data packet is knowable from its header information, hence each probe or data packet visits a particular sequence of queues as it traverses the DCN. This gives rise to the following linear equations:

$$\begin{bmatrix} D^d \\ D^p \end{bmatrix} = \begin{bmatrix} A^d \\ A^p \end{bmatrix} W + \begin{bmatrix} Z^d \\ Z^p \end{bmatrix} \tag{3}$$

where A^d and A^p are, respectively, the $N^d \times N^q$ and the $N^p \times N^q$ 0-1-valued *incidence matrices*, identifying the set of queues visited by each probe and data packet; and Z^d and Z^p model noise due to various factors such as (i) faulty NIC timestamps, (ii) differences in propagation times on cables of different lengths, (iii) variations in the switching times at different switches, etc. The queuing times are typically at least a few microseconds and typically they are 10s or 100s of microseconds. By comparison, the noise is in the order of 10–100 nanoseconds.

Setting

$$D = \begin{bmatrix} D^d \\ D^p \end{bmatrix}, A = \begin{bmatrix} A^d \\ A^p \end{bmatrix}, \text{ and } Z = \begin{bmatrix} Z^d \\ Z^p \end{bmatrix},$$

equation (3) becomes

$$D = AW + Z. \tag{4}$$

The Reconstruction. Given the vector (D^d, D^q), we are interested in getting an estimate, $\hat{W}$, of W. A natural criterion for the goodness of the estimate would be to minimize the mean squared error, $\mathbb{E}\left[(W - \hat{W})^2\right]$, where the expectation is over the noise. Typically, in a fat-tree network the number of queues with a positive delay is quite small even under high load; that is, W is typically a *sparse* vector (see [9]; we have also observed this in all our real-world experiments).

[10] Of course, probes may be dropped. We choose to ignore such probes and find that we obtain very good reconstruction results with the remaining probes, since these latter probes accurately capture large queue sizes and wait times are accurately captured from the other probes. Moreover, since probes are only 64 Bytes, they're much less likely to be dropped than data packets which are typically in the 500–1500 Bytes range. However, dropped probes convey valuable information about congestion and it is worthwhile to treat them specially. Due to a lack of space, we don't explore this aspect further in this paper.

It is not hard to show that any DCN with a multi-stage Clos topology interconnecting the servers has an adjacency matrix whose rank is less than the number of links, hence queues. This follows as a generalization of the following lemma, whose proof is simple and is omitted.

Lemma 1. *Consider all equations in the $m+n$ variables $A_1, ..., A_m$ and $B_1, ..., B_n$ of the type $A_i + B_j$. This system of equations has a maximum rank equal to $m+n-1$.*

The previous discussion shows that the rank of the matrix A is less than N^q, hence equation (4) is underdetermined. The statistical procedure LASSO [53] is naturally suited for our problem. Thus, we seek

$$\hat{W} = \arg\min_W \|D - AW\|_2^2 + \alpha\|W\|_1, \tag{5}$$

where $\alpha > 0$ is a scalar multiplying the regularization term $\|\cdot\|_p$ is the standard L_p norm. We motivate the eventual solution by proceeding through the following simpler and instructive cases.

Case 1: Consider only the data packet timestamps. Disregard the probe packets and consider the equation $D^d = A^d W + Z^d$. We generate the data using the NS setup described in Section 2.1.

We solve

$$\hat{W} = \arg\min_W \|D^d - A^d W\|_2^2 + \alpha\|W\|_1$$

and compare the solution with the actual value of W (ground truth). The comparison is shown in Figures 7 and 8 below. The solid blue line is the ground truth and the red filling is the LASSO solution. Figure 7 shows a fairly accurate reconstruction at one queue. However, Figure 8 shows the LASSO algorithm has misattributed the queue sizes amongst the 3 queues shown in the figure. Essentially, what is going on is that the equations provided by the data packets yield an underdetermined linear system (not enough equations for the variables). Therefore, not all queues are correctly reconstructed.

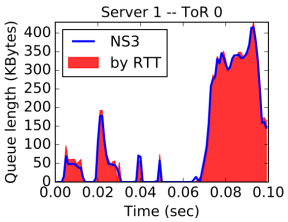


Figure 7: Reconstruction with data packets: good case

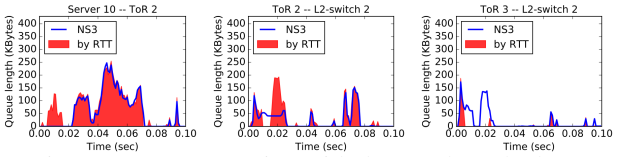


Figure 8: Reconstruction with data packets: bad case

Case 2: Consider only the probe packet timestamps. This time disregard data packets and consider the noisy linear sys-

tem

$$D^p = A^p W + Z^p. \quad (6)$$

We need to solve the inverse problem:

$$\hat{W} = \arg\min_W \|D^p - A^p W\|_2^2 + \alpha \|W\|_1.$$

The crucial difference between this case and Case 1 is that in Case 1 we cannot choose A^d but we can certainly choose A^p. We will say more about how to choose A^p in Section 3.1. For now, suppose each server probes to 10 ($K = 10$) other servers; thus, in every reconstruction interval, an average of 2×2560=5120 probes are sent.

As seen in Figure 9, the algorithm reconstructs all the queues very precisely with probe packet data. Because the probe mesh covers all the links fairly uniformly, the system of linear equations along with the sparsity constraint (captured in the L_1 regularizing term, $\|W\|_1$) determine the correct solution. Statistically, when compared with the ground truth, the algorithm achieves a root-mean-square error *across all the queues* of 5.2KB; that is, $RMS(\hat{Q} - Q) = 5.2KB$.[11]

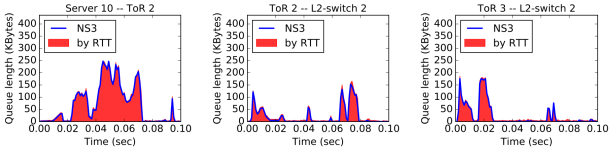


Figure 9: Network reconstruction with probe packets

3.1 Specifying the parameters of the probe mesh

As the previous reconstruction scenario has revealed, using a random graph for the probe mesh with $K = 10$ produces good reconstructions. A natural question is what is the right value of K to get a good reconstruction quality and does this depend on the DCN topology. We proceed by refining our definition of the probe mesh introduced earlier.

Definition: Probe graph. Let $G = (V, E)$ denote a graph with V equal to the servers in a given DCN, and $(i \to j) \in E$ if i probes j. G is called the *probe graph*.

Remark. Note that the above definition does not include the topology of the underlying DCN.

Definition: DCN probe mesh. Given the probe graph $G = (V, E)$ and a DCN, the *DCN probe mesh*, G_D, is the graph obtained by assigning DCN paths to probing edges $(i, j) \in E$. Note that the adjacency or incidence matrix of G_D is exactly equal to A^p defined by equation (3).

Observation 1. In order to reconstruct the waiting time or queue size of any queue in the DCN, a *necessary condition* is that queue is present on some path of G_D or, equally, that the column in A^p corresponding to that queue is not identically zero. While this condition is necessary, it is not sufficient since it doesn't guarantee solvability of equation (6).

Observation 2. The maximal condition for the solvability of equation (6) is that the rank of A^p is at its maximum. Then the LASSO algorithm along with the sparsity constraint enforced by the L_1 regularizing term will produce a unique solution.

In order to understand conditions on K which give us an A^p with full rank, we simulate a variety of different DCNs and vary K to see how that affects the *RMS* error of queue size vector, Q. Figure 10 shows that the reconstruction accuracy greatly improves with K initially and then it tapers off. More importantly, the figure suggests that this relationship holds for several different network topologies and types of queue (the layer in the DCN the queue is at).

Clearly, even though higher values of K give better reconstruction quality, there is a penalty for setting K large: servers need to issue more probes, there is the overhead of timestamping, data collection and storage, and there is the effort of reconstruction. Quantitatively, a value of $K \in [10, 20]$ seems to achieve the best reconstruction quality while representing a small enough effort.

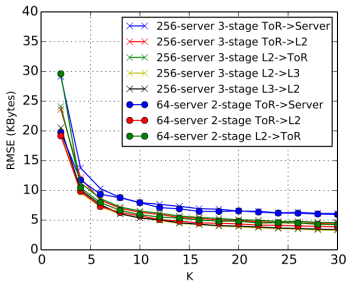


Figure 10: Reconstruction quality vs K

We provide an intuitive explanation of why a constant value of $K \in [10, 20]$ is sufficient to reconstruct the queues regardless of the size of the network, and give a probabilistic argument on the queue coverage probability in the appendix. As the size of network grows, the number of servers grows, hence the total number of probes will grow (for a fixed K), resulting in a constant sampling rate of each queue. To quantify the foregoing under a "full bisection bandwidth assumption" on the DCN, suppose we have N servers in the DCN. If we cut the network into two sets of servers, S_1 and S_2, respectively with N_1 and N_2 servers (WLOG we assume $N_1 \leq N_2$), then the total number of links crossing the boundary of the two sections is N_1 (because of full bisection bandwidth). Since each server probes K servers chosen at random, the expected number of probes and echoes passing these $N1$ links from S_1 to S_2 and from S_2 to S_1 each equal $\frac{2N_1N_2K}{N}$. Thus, every queue on the boundary is sampled on average by $\frac{2N_2K}{N} \geq K$ probes. Therefore, as long as $K \geq 10$, we can guarantee each queue is sampled by at least 10 probes.

[11] Note that we use the standard definition of *RMS*; i.e., $RMS(Q) = \|Q\|_2$. We choose to show the reconstruction plots and report the errors for the queue size vector Q rather than for the delay vector W since we have found that the queue sizes are usually in the order of a few 100 KBs regardless of the line rate whereas the waiting times vary with the line rate.

A final observation about Figure 10: the RMSE appears to drop like $\frac{1}{\sqrt{K}}$. Under the assumption that each probe only visits one congested queue (recall that DCN congestion is sparse), this can be explained as follows. If K probes pass through a queue with a reconstruction-interval-average size of Q and make independent noisy observations with variance σ^2 (where $\sigma^2 \approx (12KB)^2$), then the RMSE roughly equals $\sigma/\sqrt{K}$, which for $\sigma = 12KB$ and $K = 10$ approximately equals 4KB, as seen in Figure 10.

3.2 Link utilization and queue/link compositions

We're now interested in solving for all the performance measures, not just queue sizes or wait times. In particular, probes can only detect positive wait times; however, zero wait times don't imply absence of traffic! The link utilization may be too small to cause queueing delays. In order to determine link utilizations and the composition of queues and links (i.e., breaking down queue and link occupancies in a reconstruction interval according to flow ids), we have to use both the probes and data packets. We explain how to do this algorithmically, following the steps below.

Step 1. Solve Case 2 and obtain the (reconstruction interval average) queuing sizes and wait times, $(Q,W) = (q_k, w_k), 1 \le k \le N^q$.

Step 2. Use W to determine the position of a probe or data packet at each instant from transmission to reception. As an example, consider probe i, transmitted at time t_i^p and received at time r_i^p. Suppose this probe visited queues with delays $w_{k_1}, w_{k_2}, ..., w_{k_L}$. Note that $r_i^p - t_i^p \approx w_{k_1} + w_{k_2} + ... + w_{k_L} + P$, where P represents the total time probe i spent on all the links on the path. At time $t \in (t_i^p + w_{k_1}, t_i^p + w_{k_1} + w_{k_2})$ we assert that probe i is either in the second queue on its path or on the link connecting the first and second queues depending on how much larger than $t_i^p + w_{k_1}$ is t. We can also similarly determine the position of each data packet. Note that extra care must be taken to account for the time spent by a data packet in a queue or on a link since its length can be much larger than that of the probe packets.

In this manner we can decompose the contents of a queue into traffic segments of interest and specified by headers. We can similarly also obtain the link utilizations and link compositions.

To see how well the above procedure works, we again look at the NS simulation setup from Section 2.1. Figure 11 shows some examples of queue and link utilization compositions, comparing them to the ground truth taken from ns-3. There are three sets of plots, left, middle and right. Each set has two plots: a queue size plot on top and a link utilization plot on the bottom. The figure on the left shows the utilization of the link connecting L2-switch 8 to L3-switch 0 on the bottom and the size of the queue attached to this link on top. We see the reconstructed queue occupancies of the blue, yellow and green packets shown by the filling and the corresponding ground truth shown by the black lines. Here the blue, yellow and green packets are destined for servers in racks 0-7, 16-23 and 24-31, respectively. It is worth noting that when the link utilization nears 100%, the size of the corresponding queue shoots up; otherwise, when the link utilization is well below 100%, there is no queue buildup. The other two sets of plots show other links.

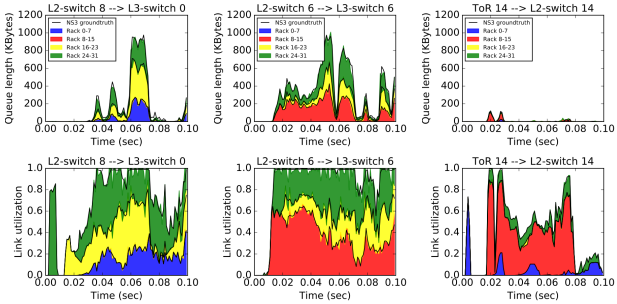


Figure 11: Reconstructing queue size and link utilization compositions

As can be seen, the SIMON algorithm does an excellent job of determining the queue size and link utilization compositions. Statistically, the reconstruction precisions (RMSE) of the queue size and link utilization composition for each class of traffic are 4.14KB and 1.01% (i.e. 0.101 Gbps in 10Gbps net), respectively.

Method	RMSE of composition		Storage space	Run time[12]
	Queue	Link util		
Per packet	4.14KB	1.01%	2.84MB	4.00ms
100us count	4.12KB	0.98%	172.3KB	0.277ms
250us count	4.08KB	0.94%	98.4KB	0.173ms
500us count	4.19KB	1.01%	69.4KB	0.142ms
1ms count	4.71KB	1.81%	49.5KB	0.107ms

Table 1: Use byte counts to decompose queue and link utilization. Space and time are for 1 reconstruction interval.

The effort—space and computation time—needed for the link utilization and queue/link compositions can be vastly simplified if we count the number of Bytes sent by each 5-tuple in Step 2 of the algorithm rather than take the transmit timestamps for each data packet. We increase the interval over which the byte counts are taken from 100 us to 1 ms while tracking the reconstruction quality. The results are shown in Table 1. As can be seen in the table, using byte counts over a 500us interval, the space consumption is reduced by 41x and the run time is reduced by 28x with almost no change to the reconstruction quality.

[12] C++ single-thread program on an Intel Core i7 processor

4 Speeding up the implementation of SIMON

The LASSO algorithm optimizes a convex cost function over a large number of variables using gradient descent. Because it is iterative, for large problem instances, LASSO can take time. Even though reconstruction can be done offline and still be quite valuable, it is interesting to ask if reconstruction can be done in near-real time and, if so, how this would be possible. This would enable rapidly detecting bottlenecks and anomalies, raising alerts, as well as enabling sensitive A/B tests in near-real time.

We explore two ideas to accelerate SIMON in large networks. Section 4.1 decomposes the reconstruction problem to a hierarchy of subnets that can be solved individually and in parallel. Section 4.2 uses neural networks to function approximate LASSO and obtain significant speed improvements. These two methods allow us to scale SIMON to large data centers with tens of thousands of servers.

4.1 Hierarchical reconstruction

The fat-tree has become the de facto data center networking topology [5, 47]. The hierarchical nature of the fat-tree topology allows us to propose an algorithm that groups queues in layers and perform reconstruction from lower layers to upper layers. The reconstruction for each layer may contain a number of sub-reconstructions that can run in parrallel. After queues at lower network layers are reconstructed, their sizes will be used to reconstruct higher-level queues.

As an example, we illustrate the hierarchical reconstruction algorithm using the topology in Figure 2. For notational simplicity, we use *Li blocks* to denote the forest after truncating the tree at layer *i*; that is, the connected components of the subgraph containing only switches of layer *i* or lower, and the servers. In Figure 2, there are 16 L1 blocks, 4 L2 blocks and 1 L3 block. We also define L*i* probes to be the probes whose source and destination are within the same L*i* block. The procedure is as follows:

1. We extract the equations given by the L1 probes. Each L1 block (rack) provides a system of linear equations, so there will be in total 16 of them. By solving the 16 linear systems, we reconstruct all queues in the L1 blocks (downlink of ToR switches).

2. We then reconstruct queues in the L2 block—the ToR uplink queues and the L2 switch downlink queues—using L2 probes. Each L2 probe passes through 3 queues: one ToR uplink, one L2 downlink and one ToR downlink. Since the ToR downlinks have been solved in the previous step, each L2 probe now provides an linear equation of the other two remaining queues. There are four linear systems for every L2 block. The tier 2 queues can then be reconstructed by solving the 4 linear systems.

3. Lastly, we reconstruct the L3 queues with L3 probes, where we repeat the previous step. After solving L3 queues, all queues in the network would have been reconstructed.

#servers	#network layers	LASSO		Hierarchical LASSO	
		RMSE	Run Time	RMSE	Run Time
256	3	5.22KB	37.34ms	9.47KB	0.59ms
2048	4	5.76KB	532.60ms	10.3KB	1.25ms
4096	4	5.53KB	1147.22ms	10.69KB	1.75ms

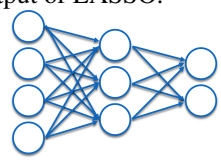
Table 2: RMSE and Run Time Comparison

Performance. Table 2 show that LASSO can be sped up by several orders of magnitude, especially as the network size increases with a modest sacrifice of accuracy.

4.2 Accelerating reconstruction with neural networks

In this section we explore using the function approximation [29] capability of multi-layered neural networks to speed up LASSO and rapidly estimate the vector of queue sizes, $\hat{Q}$, from the vector of end-to-end probe delays, D. The goal is to function-approximate LASSO and learn the underlying function mapping D to $\hat{Q}$ from the training examples.[13] Since the neural network only involves simple arithmetic operations which can be implemented at high speed on modern hardwares such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs), we can potentially get tremendous speedups.

We take 60 seconds of data at 1 ms intervals from the NS scenario. The data are 60,000 (D, Q) pairs where each D is a 5120-dimensional vector and each Q is a 1280-dimensional vector (to be more specific, we consider D^p and not D^d; for ease of notation, we shall denote D^p as D). We also use LASSO's inferred queue sizes, $\hat{Q}$, following the reconstruction procedure in Section 3 and use 60,000 $(D, \hat{Q})$ pairs to train the neural network. This allows us to compare two training methods: one with the ground truth from ns-3 and the other is the output of LASSO.



$$D \in \mathbb{R}^{\text{input}} \quad H \in \mathbb{R}^{\text{hidden}} \quad \hat{Q} \in \mathbb{R}^{\text{output}}$$
$$\text{input} = |D|, \text{output} = |\hat{Q}|$$
$$\hat{Q} = M \times ReLU(LD + b) + l$$

Figure 12: Neural network for network reconstruction

We use these two sets of data to train two instances of a ReLU (Rectified Linear Unit [39]) neural network with just 1 hidden layer, as shown in Figure 12. At the end of the training period, each neural network has "learned" the matrices

[13] Note that LASSO is a general statistical procedure; the fact that it may be function-approximable in this context is not an indication that it would work in other cases. Our estimation problem is special: we're solving a system of noisy linear equations where the noise is additive. It may be harder to function approximate LASSO in nonlinear systems and/or with multiplicative noise.

L and M and the bias vectors b and l from their respective input-output pairs. We then test each neural network with new values of D to see how well the corresponding output agrees with the ground truth Q. The results are shown in Table 3. As can be seen, both neural networks perform very well when compared to LASSO; indeed, the neural network trained with the ground truth does better than LASSO.

RMSE (Bytes)	LASSO	Neural Net (trained w. GT)	Neural Net (trained w. LASSO)
256 server	5.22KB	4.15KB	5.34KB
2048 server, 4 stage	5.76KB	4.03KB	7.53KB
4096 server, 4 stage	5.53KB	3.95KB	7.08KB

Table 3: RMSE Comparison

Table 4 compares the speed of the neural networks with that of LASSO. The profiling of LASSO was run with Python Scikit-learn sparse LASSO package, on a server with 6-core Intel I7 processor, and the neural network profiling are done on an Nvidia GTX1080 GPU using Tensorflow[14]. The neural network is three magnitudes faster than LASSO during inference time due to the algebraic-simplicity of its forward pass and speed-up with hardware accelerators.

Run Time	LASSO (single core)	LASSO (6-core)	Neural Net
256 server	37.34ms	6.22ms	2.75us
2048 server, 4 stage	532.60ms	96.14ms	76.24us
4096 server, 4 stage	1147.22ms	208.25ms	235.97us

Table 4: Speed Comparison

There are two major points to make about using neural networks for data center measurement.

The first is to note that whereas LASSO requires the adjacency matrix A^q, the neural networks don't require it. This point and the abundance of operational data in modern data centers strongly suggest that neural networks can play a major role as a lightweight, very high speed system for detecting anomalies and even accurately estimating key performance measures.

For the second, consider the scenario where the network topology changes or the ECMP hash functions change (e.g., due to link failures) in the network. In modern DCNs that support software-defined networking (SDN), such changes can be known when the SDN controller receives corresponding events. In these cases, the D, Q relationship would change, which could make the already trained neural network obsolete. Thus, it may be required to re-train the neural network. However, naively employing neural networks for large data centers implies very long re-training time. Taking the 4096-server case as an example, even though training the neural network only requires 288 seconds worth of timestamp data, it takes 8 hours for the weights in the neural network to converge.

[14]We used the best off-the-shelf implementation we can find for both LASSO and NN. Our effort to speed up LASSO with GPUs using the SHOTGUN algorithm did not result in much gain. LASSO iteratively optimizes a convex cost function using gradient descent, a task better-suited for CPUs than for GPUs.

In practice, we would use neural networks in combination with hierarchical reconstruction. This would not only speed up retraining (because subnets can be retrained in parallel), but it would also make the whole process more robust to failures (since only subnets affected by failure need to be retrained). For example, retraining a 256-server subnet only requires ten seconds worth of $(D, \hat{Q})$ pairs and converges in 2.5 minutes on a Nvidia GTX1080 GPU.

5 Implementation

We implement the LASSO version[15] of SIMON for Linux. This implementation has three components: the prober, the sensor and the reconstruction engine. The prober is implemented as a user space program. There is one prober sitting on each of the servers in the network. Each of the probers repeatedly probes K other random probers and echoes received probes. The sensor is implemented in the NIC driver and is in charge of collecting hardware transmit and receive timestamps for both data packets and probes. Then, upon reconstruction requests, the sensors will batch the time stamp data for the requested time range and ship the data to the reconstruction engine.

The reconstruction engine is a Spark Streaming [48] cluster. It takes timestamp data over a given time range as input and outputs the reconstructed queue lengths, link utilizations and their per-flow break downs for each reconstruction interval in that time range. The reconstruction engine exploits the fact that the data processing for different reconstruction intervals are independent. Upon the arrival of the timestamp data at the reconstruction engine, the data is originally sharded by server IDs since individual servers collected the data. Then the engine will re-shard the data into reconstruction intervals, and assign each interval to one of the servers in the cluster. Finally, after the reconstruction for each interval is done, the results are stitched together and returned to the user.

6 Deployment and Validation

This section uses three very different testbeds to verify that SIMON works with a wide variety of network configurations and under real-world complexities. The testbeds have link speeds ranging from 1 Gbps to 40 Gbps, contain 2 to 5 stages of switching fabric, and employ switches from different vendors and models. Furthermore, we discuss some pragmatic experience gained from these deployments, namely, how to still output meaningful reconstruction results with only partial probing coverage of the network. We use NetFPGAs to verify the accuracy of SIMON in the 1 Gbps testbed. Finally, we use cross-validation to verify and evaluate the correctness of the reconstruction results in the absence of ground

[15]Hierarchical reconstruction and neural networks are planned to be implemented for production environments in the future.

truth data.

We first specify the configuration of the three testbeds.

Testbed T-40G-3L. This is a 3-stage Clos network, all links are 40Gbps. It has 20 racks with roughly 12 servers per rack. There are 32 switches at both spine layers.

Testbed T-10G-40G-5L. This is a 5-stage Clos network containing ∼500 racks, each of which has ∼24 servers. Each server is connected to a ToR switch with two 10G aggregated links. The rest of the links are all 40G. We had access to 12 out of the 500 racks.

Testbed T-1G-2L. This is a 2-stage Clos network with all 1 Gbps links. It consists of 8 racks, each rack has 16 servers. There are 8 spine switches. The switches in the testbed are Cisco 2960s and Cisco 3560s.

6.1 Deployment experience

SIMON makes minimal assumptions about the network and only relies on data sensed at the NICs, hence its deployment has generally been smooth. SIMON needs two things: (i) hardware timestamping-capable NICs and (ii) the ability to know the paths taken by the packets. For (i), almost all current generation 10G or above NICs can timestamp all ingress and egress packets. In T-1G-2L, even the on-board 1G NICs support it. For (ii), since per-flow ECMP is the standard load-balancing scheme used by the industry, knowing the paths taken by the packets is not difficult. In T-40G-3L and T-10G-40G-5L, we ran traceroute [36] periodically to learn the paths taken by any targeted flows. In T-1G-2L, since we use 5-tuple-based source routing, we automatically know the paths taken by any packets basing on their 5-tuples.

Despite the overall smoothness of the deployments, one case that did require some extra care was when two or more queues cannot be further disambiguated by the probes. For example, in our deployment in T-10G-40G-5L, since we only had access to a small portion of the racks in a big network, our probes can only reach a subset of all the queues in the network. Within these reachable queues, some of them are always bundled together as seen by the probes. That is, a probe either traverses all of these queues or none of them. In this case, we treat the sum of these queues as a single virtual queue, and treat this virtual queue as a single variable in Equation 3.

6.2 Validating accuracy with NetFPGAs

We use NetFPGAs to validate the precision of SIMON in T-1G-2L. Figure 13 shows the setup. We configure the four ports in a NetFPGA, P_0, P_1, P_2, P_3, as two pass-through timestampers: P_0 and P_1 are connected back-to-back via the FPGA, and packets passing through these two ports will be timestamped according to a local counter in the FPGA; P_2 and P_3 are similar except the timestamp is inserted into the packet at a different offset. We connect the two timestampers to two different ports of the same switch, S_0 and S_1. For all packets between S_0 and S_1, we will know their overall switching and queueing delays in the switch from the NetFPGA timestamps. After subtracting the switching delay which can be measured when the network is idle, the remaining packet queueing delays can serve as ground truth for queueing delays of the output queues behind S_0 and S_1.

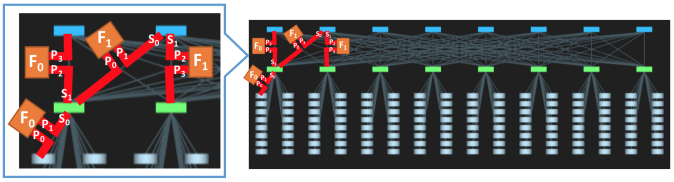
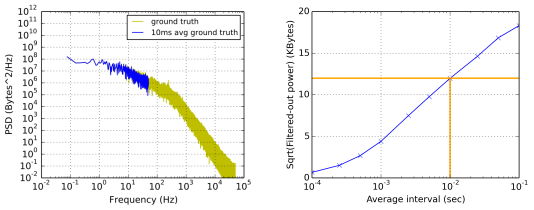


Figure 13: Setup of reconstruction validation in T-1G-2L. F_0 and F_1 are two NetFPGAs.



(a) PSD (b) Filtered-out power

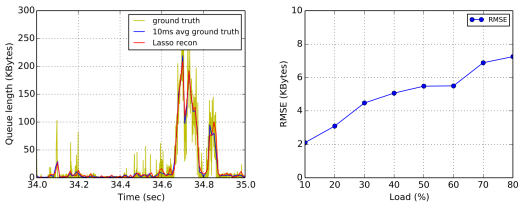
Figure 14: PSD of queues with NetFPGA ground truth at 40% load

An incast load (same as in Section 3) is applied with loads between ToR switches and spine switches ranging from 10% to 80%. Each server probes 20 other servers ($K = 20$). The 10 ms average queue depths are reconstructed using probe timestamps with LASSO. Separately, we send extra probes through the queues enclosed by NetFPGAs to collect the ground truth waiting time. The reconstruction results are compared to the ground truth.

Figure 14 shows the PSDs of the observed queueing processes. It shows that the power of the filtered-out high frequency component is $(12KB)^2$, which matches the results in Section 3! Figure 15 shows the reconstruction results of SIMON verified against ground truth. We can see that SIMON accurately reconstructs the queue depths for different levels of switches at various network loads.

6.3 Cross-validation

It was not possible to get accurate ground truth in the testbeds T-40G-3L and T-10G-40G-5L. Most switches only support reading counter samples a few tens of times per second, and these samples are not taken frequently enough and cannot be otherwise used to get statistical quantities like millisecond-average queue sizes. We also did not have physical access to these data centers, hence NetFPGA-based evaluations were not possible.



(a) Recon at 40% load (b) Errors at different loads

Figure 15: Reconstruction compared with NetFPGA ground truth

We proceed by using cross-validation to verify the accuracy of reconstruction. Consider two sets of probes, blue probes and red probes. No blue probing pair of servers is common to a red probing pair, nor is any end-to-end DCN path common to the blue and red probes: the two sets of probes are completely non-overlapping. Let the delay vectors and the incidence matrices for the blue and red probes be D_{blue}, D_{red}, A_{blue} and A_{red}, respectively. If, using the blue probes, we can get a reconstruction of the queueing delays $\hat{W}_{blue}$, and use this to accurately "predict" the delays of the red probes as $\hat{D}_{red} = A_{red}\hat{W}_{blue}$, then we can get a validation of the accuracy of our reconstruction.

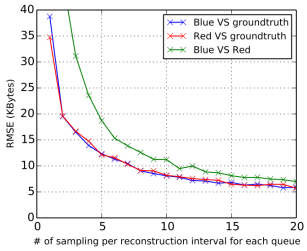


Figure 16: Blue red validation in simulations

Here we present a variant of the above method: instead of testing the agreement between $\hat{D}_{red}$ and D_{red}, we test the agreement between $\hat{Q}_{blue}$ and the reconstruction from red probes, $\hat{Q}_{red}$. We use this method because it is better for presentation (we get to plot the queue depths) and it helps bound the quality of the reconstruction: Denote the reconstruction error of the blue and red probes as $N_{blue} = \hat{Q}_{blue} - Q$ and $N_{red} = \hat{Q}_{red} - Q$ respectively. Assuming N_{blue} and N_{red} have zero mean and are independent, we have $\mathbb{E}\left[(\hat{Q}_{blue} - \hat{Q}_{red})^2\right] = \mathbb{E}\left[N_{blue}^2\right] + \mathbb{E}\left[N_{red}^2\right]$, which means the difference between the two reconstructions is bigger than the error of either reconstruction. Figure 16 illustrates this empirically with ns-3 simulations. As can be seen, at different queue sampling rates (probing rates), the error between the reconstructions obtained from the blue and red probes upper bonds their respective reconstruction errors (with respect to the ground truth).

This verification was performed on all three testbeds; we consider T-40G-3L. Each server in T-40G-3L blue probes to 10 random destination servers, and sends red probes to 10 *other* random destination servers. Since the links are all 40G, the reconstruction interval is 250μs. Figure 17 shows the reconstruction results at two different queues during a 100ms interval. Each sub-figure shows the blue-red comparison at one queue, with the upper half showing the blue reconstruction and the lower half showing the red probe one. There is excellent agreement between the blue and red reconstructions.

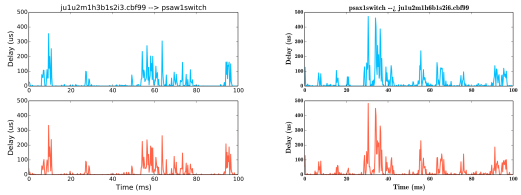


Figure 17: Blue-red validation examples in T-40G-3L

6.4 Example use cases of SIMON.

SIMON gives network administrators and application developers detailed visibility into the state and the dynamic evolution of the network. Thus, SIMON has many use cases in network regression testing, A/B testing, anomaly detection and bottleneck localization. We share a few of our experiences.

For the scenario in Figure 17, the normal maximum delay in a single queue is around 500μs in T-40G-3L. However, during an experiment, SIMON showed that the queueing delays in T-40G-3L suddenly increased to 5 to 10 milliseconds (shown in Figure 18), which is improbable in a 40G network as this implies very large buffers. This puzzling behavior was quickly resolved by observing the delays rose vertically in Figure 18 and that no packets were dropped. The only explanation is that the switches were configured to strict priority scheduling and our traffic was blocked by some higher priority traffic. This turned out to be the case. Thus, SIMON not only surfaced and explained the anomaly in the network, but also helped understand the precise way strict priority affects low-priority traffic.

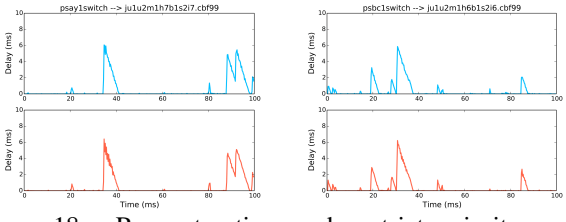


Figure 18: Reconstruction under strict priority packet scheduling in T-40G-3L

Another case: In T-1G-2L, SIMON discovered that one of the 256 1 Gbps ports in the testbed was mistakenly configured to 100 Mbps—a fact that is hard to determine without a

detailed link utilization plot! Moreover, in the same testbed, SIMON found that one application's traffic was frequently causing other applications' TCP flows to timeout. Many of these uses cases are beyond the reach of traditional network monitoring tools. We believe the visibility provided by SIMON can greatly increase the ability to understand and efficiently utilize the network.

7 Literature Survey

Switch-based methods. There are three main threads: (a) Obtain summaries and sketches of large, high-rate flows (called "heavy hitters") as these are easy to identify and account for most of the bandwidth consumption and queue backlogs [23, 43, 35, 15]. (b) Capture detailed counts of *all packets and all flows*; for example, [46, 45] uses a hybrid SRAM-DRAM memory architecture and [34] uses CounterBraids to "compress while counting". These methods are harder to implement compared to those in (a) and entail offline processing to obtain the exact counts. (c) In-band network telemetry (INT) [58, 30, 32] uses switches to append telemetry data to packet headers as they pass through. The "postcards" approach [28] gathers telemetry data for all packets and sends it together with packet headers to servers. This approach takes advantage of the flexibility in P4 programmable switches and the storage and computation resources in servers, and is the focus of much current research.
Edge-based methods. Trumpet [38] uses flow event triggers to detect flow-level traffic spikes and congestion; for every packet dropped, 007 [8] attempts to identify the link responsible for the drop knowing the path taken by the packet (from traceroute); Pingmesh [27] uses RTT measurements of TCP probes to infer end-to-end latency distributions between any pair of hosts, including "packet black holes"; PathDump [51] uses switches to tag packets belonging to flows that satisfy a certain criterion and processes and stores the data at the edge; and SwitchPointer [52] goes further: it uses switches to store pointers to telemetry data relevant to that switch but held in end-hosts. PathDump and SwitchPointer follow the work of Everflow [59] which makes significant use of switches to "match" specific packets and "mirror" them to servers which can then trace the packets across the whole path. This enables Everflow to detect various faults such as silent packet drops, routing loops and load imbalance.
Tomography. As mentioned in the Introduction, reconstruction of network state variables in the wide area is not possible due to the long propagation times, unknown topology, and path information. Hence, researchers have obtained *distributions* of delays and backlogs of [42, 21, 19, 56, 55, 57, 27], topological relationships [17], or packet loss patterns at switch buffers [12, 11, 22, 54, 18, 26, 27].
In summary, switch-based methods are not easy to scale and can't easily relate network bottlenecks to application performance since they lack application context. Current edge-based methods obtain only a partial view of the network state and some may require non-trivial assistance from switches. Tomography results in the wide-area do not obtain near-exact reconstruction of network variables. To our knowledge, this is the first application of Network Tomography to data centers and for obtaining a near-exact reconstruction.

8 Conclusion

We introduced SIMON, a network tomography-based sensing, inference and measurement system for data centers. SIMON reconstructs network state variables, such as queueing times, link utilizations, and queue and link compositions, using timestamps of data packets and probes taken at NICs. By doing so, SIMON is able to connect bottlenecks at switches and network links to individual flows of the applications.
SIMON employed several techniques to simultaneously achieve precise reconstruction and scalability. First, a signal processing analysis suggested that reconstruction intervals that vary inversely as link speeds—10 ms for 1 Gbps links, 1 ms for 10 Gbps links, 250 μs for 40 Gbps links, are appropriate for reconstructing queue sizes and wait times to an accuracy of over 97% of the power of the queue size or wait time process. Secondly, we used a mesh of probes to obtain extra information about network queue sizes and wait times, and described guidelines for picking the parameters of the probe mesh. We then showed that a LASSO-based reconstruction procedure accurately reconstructed the queue size, wait time, link utilization and queue/link composition processes. Two methods of simplifying the implementation of LASSO were presented: one method exploited the hierarchical structure of modern data center topologies to get a 1000x speedup of SIMON, and the other method used multi-layered neural networks and GPUs to accelerate LASSO by 5,000x–10,000x. These methods enable SIMON to run in near real-time.
We implemented SIMON on three testbeds with link speeds ranging from 1–40 Gbps and with up to 288 servers. The accuracy of the reconstruction is validated with NetFPGAs in the 1 Gbps testbed and by a cross validation technique in the other two testbeds. These implementations not only demonstrate the accuracy of SIMON's reconstruction but also demonstrated its practicability. Since SIMON is agnostic of the switches in the network and only requires the timestamping capability readily available in most current generation NICs, it can be deployed in production data centers today.

References

[1] Appdynamics, 2017. [Online; accessed 9-October-2017].

[2] New relic, 2017. [Online; accessed 9-October-2017].

[3] Splunk, 2017. [Online; accessed 9-October-2017].

[4] Cisco tetration, 2018. [Online; accessed 14-February-2018].

[5] AL-FARES, M., LOUKISSAS, A., AND VAHDAT, A. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review* (2008), vol. 38, ACM, pp. 63–74.

[6] ALIZADEH, M., GREENBERG, A., MALTZ, D. A., PADHYE, J., PATEL, P., PRABHAKAR, B., SENGUPTA, S., AND SRIDHARAN, M. Data center tcp (dctcp). In *ACM SIGCOMM computer communication review* (2010), vol. 40, ACM, pp. 63–74.

[7] ARISTA. Eos cloud networking technology - telemetry and analytics, 2018. [Online; accessed 9-January-2018].

[8] ARZANI, B., CIRACI, S., CHAMON, L., ZHU, Y., LIU, H., PADHYE, J., LOO, B. T., AND OUTHRED, G. 007: Democratically finding the cause of packet drops. In *NSDI* (2018), USENIX.

[9] BENSON, T., AKELLA, A., AND MALTZ, D. A. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement* (2010), ACM, pp. 267–280.

[10] BROADCOM. High-performance data center soc with integrated netxtreme ethernet controller, 2018. [Online; accessed 9-October-2017].

[11] BU, T., DUFFIELD, N., PRESTI, F. L., AND TOWSLEY, D. Network tomography on general topologies. In *ACM SIGMETRICS Performance Evaluation Review* (2002), vol. 30, ACM, pp. 21–30.

[12] CÁCERES, R., DUFFIELD, N. G., HOROWITZ, J., AND TOWSLEY, D. F. Multicast-based inference of network-internal loss characteristics. *IEEE Transactions on Information theory 45*, 7 (1999), 2462–2480.

[13] CASE, J. D., FEDOR, M., SCHOFFSTALL, M. L., AND DAVIN, J. Simple network management protocol (snmp). Tech. rep., 1990.

[14] CAULFIELD, A. M., CHUNG, E. S., PUTNAM, A., ANGEPAT, H., FOWERS, J., HASELMAN, M., HEIL, S., HUMPHREY, M., KAUR, P., KIM, J.-Y., ET AL. A cloud-scale acceleration architecture. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on* (2016), IEEE, pp. 1–13.

[15] CHARIKAR, M., CHEN, K., AND FARACH-COLTON, M. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming* (2002), Springer, pp. 693–703.

[16] CHUANG, S.-T., GOEL, A., MCKEOWN, N., AND PRABHAKAR, B. Matching output queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications 17*, 6 (1999), 1030–1039.

[17] COATES, M., CASTRO, R., NOWAK, R., GADHIOK, M., KING, R., AND TSANG, Y. Maximum likelihood network topology identification from edge-based unicast measurements. In *ACM SIGMETRICS Performance Evaluation Review* (2002), vol. 30, ACM, pp. 11–20.

[18] COATES, M. J., AND NOWAK, R. D. Network loss inference using unicast end-to-end measurement. In *ITC Conference on IP Traffic, Modeling and Management* (2000), pp. 28–1.

[19] COATES, M. J., AND NOWAK, R. D. Network tomography for internal delay estimation. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on* (2001), vol. 6, IEEE, pp. 3409–3412.

[20] CONTRIBUTORS, W. Syslog — wikipedia, the free encyclopedia, 2018. [Online; accessed 9-January-2018].

[21] DUFFIELD, N. G., HOROWITZ, J., PRESTI, F. L., AND TOWSLEY, D. Network delay tomography from end-to-end unicast measurements. In *Thyrrhenian Internatinal Workshop on Digital Communications* (2001), Springer, pp. 576–595.

[22] DUFFIELD, N. G., PRESTI, F. L., PAXSON, V., AND TOWSLEY, D. Inferring link loss using striped unicast probes. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2001), vol. 2, IEEE, pp. 915–923.

[23] ESTAN, C., AND VARGHESE, G. *New directions in traffic measurement and accounting*, vol. 32. ACM, 2002.

[24] GENG, Y., LIU, S., YIN, Z., NAIK, A., PRABHAKAR, B., ROSENBLUM, M., AND VAHDAT, A. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), pp. 81–94.

[25] GHITA, D., KARAKUS, C., ARGYRAKI, K., AND THIRAN, P. Shifting network tomography toward a practical goal. In *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies* (2011), ACM, p. 24.

[26] GHITA, D., NGUYEN, H., KURANT, M., ARGYRAKI, K., AND THIRAN, P. Netscope: Practical network loss tomography. In *INFOCOM, 2010 Proceedings IEEE* (2010), IEEE, pp. 1–9.

[27] GUO, C., YUAN, L., XIANG, D., DANG, Y., HUANG, R., MALTZ, D., LIU, Z., WANG, V., PANG, B., CHEN, H., ET AL. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 139–152.

[28] HANDIGOL, N., HELLER, B., JEYAKUMAR, V., MAZIÈRES, D., AND MCKEOWN, N. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *NSDI* (2014), vol. 14, pp. 71–85.

[29] HAYKIN, S., AND NETWORK, N. A comprehensive foundation. *Neural networks 2*, 2004 (2004), 41.

[30] HYUN, J., AND HONG, J. W.-K. Knowledge-defined networking using in-band network telemetry. In *Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific* (2017), IEEE, pp. 54–57.

[31] IEEE. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. *IEEE Standard 1588* (2008).

[32] KIM, C., SIVARAMAN, A., KATTA, N., BAS, A., DIXIT, A., AND WOBKER, L. J. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM* (2015).

[33] LEE, K. S., WANG, H., SHRIVASTAV, V., AND WEATHERSPOON, H. Globally synchronized time via datacenter networks. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference* (2016), ACM, pp. 454–467.

[34] LU, Y., MONTANARI, A., PRABHAKAR, B., DHARMAPURIKAR, S., AND KABBANI, A. Counter braids: a novel counter architecture for per-flow measurement. *ACM SIGMETRICS Performance Evaluation Review 36*, 1 (2008), 121–132.

[35] LU, Y., PRABHAKAR, B., AND BONOMI, F. Elephanttrap: A low cost device for identifying large flows. In *High-Performance Interconnects, 2007. HOTI 2007. 15th Annual IEEE Symposium on* (2007), IEEE, pp. 99–108.

[36] MANUAL, L. traceroute manual page, 2018. [Online; accessed 9-January-2018].

[37] MELLANOX. Bluefield multicore system on chip, 2017. [Online; accessed 19-February-2018].

[38] MOSHREF, M., YU, M., GOVINDAN, R., AND VAHDAT, A. Trumpet: Timely and precise triggers in data centers. In *Proceedings of the 2016 ACM SIGCOMM Conference* (2016), ACM, pp. 129–143.

[39] NAIR, V., AND HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (2010), pp. 807–814.

[40] NETWORKS, J. Junos telemetry interface feature guide, 2018. [Online; accessed 9-January-2018].

[41] OPPENHEIM, A. V. *Discrete-time signal processing*. Pearson Education India, 1999.

[42] PRESTI, F. L., DUFFIELD, N. G., HOROWITZ, J., AND TOWSLEY, D. Multicast-based inference of network-internal delay distributions. *IEEE/ACM Transactions On Networking 10*, 6 (2002), 761–775.

[43] RAMABHADRAN, S., AND VARGHESE, G. Efficient implementation of a statistics counter architecture. In *ACM SIGMETRICS Performance Evaluation Review* (2003), vol. 31, ACM, pp. 261–271.

[44] SFLOW.ORG. sflow, 2018. [Online; accessed 9-January-2018].

[45] SHAH, D., IYER, S., PRABHAKAR, B., AND MCKEOWN, N. Analysis of a statistics counter architecture. In *Hot Interconnects 9, 2001.* (2001), IEEE, pp. 107–111.

[46] SHAH, D., IYER, S., PRAHHAKAR, B., AND MCKEOWN, N. Maintaining statistics counters in router line cards. *IEEE Micro 22*, 1 (2002), 76–81.

[47] SINGH, A., ONG, J., AGARWAL, A., ANDERSON, G., ARMISTEAD, A., BANNON, R., BOVING, S., DESAI, G., FELDERMAN, B., GERMANO, P., ET AL. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM Computer Communication Review 45*, 4 (2015), 183–197.

[48] SPARK. Spark: Lightning-fast cluster computing, 2018. [Online; accessed 9-January-2018].

[49] SYSTEMS, C. Model-driven telemetry, 2018. [Online; accessed 9-January-2018].

[50] SYSTEMS, C. Netflow, 2018. [Online; accessed 9-January-2018].

[51] TAMMANA, P., AGARWAL, R., AND LEE, M. Simplifying datacenter network debugging with pathdump. In *OSDI* (2016), USENIX.

[52] TAMMANA, P., AGARWAL, R., AND LEE, M. Distributed network monitoring and debugging with switchpointer. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)* (2018), USENIX.

[53] TIBSHIRANI, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.

[54] TSANG, Y., COATES, M., AND NOWAK, R. Passive network tomography using em algorithms. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on* (2001), vol. 3, IEEE, pp. 1469–1472.

[55] TSANG, Y., COATES, M., AND NOWAK, R. Nonparametric internet tomography. In *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on* (2002), vol. 2, IEEE, pp. II–2045.

[56] TSANG, Y., COATES, M., AND NOWAK, R. D. Network delay tomography. *IEEE Transactions on Signal Processing 51*, 8 (2003), 2125–2136.

[57] TSANG, Y., YILDIZ, M., BARFORD, P., AND NOWAK, R. Network radar: tomography from round trip time measurements. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement* (2004), ACM, pp. 175–180.

[58] VAN TU, N., HYUN, J., AND HONG, J. W.-K. Towards onos-based sdn monitoring using in-band network telemetry. In *Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific* (2017), IEEE, pp. 76–81.

[59] ZHU, Y., KANG, N., CAO, J., GREENBERG, A., LU, G., MAHAJAN, R., MALTZ, D., YUAN, L., ZHANG, M., ZHAO, B. Y., ET AL. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 479–491.

9 Appendix

9.1 Queue Coverage Probability

Consider a network with l layers and N servers. Each server randomly probes K other servers. We show that without under-subscription, *i.e.*, the number of links at the spine layers is no greater than the number of servers, the probability that every queue in the network is covered by at least one probe packet is greater than $1-(2l-1)Ne^{-2K}$, where l is the number of layers, N is the number of servers and K is the number of random probes per server.

Proof:

A lower bound on the probability can be derived using a union bound. We consider queues at different levels, and bound the probability of covering all queues at every level. First, consider the lowest level queues, the queues on the ToR downlinks to the servers. There are exactly $N_1 = N$ such queues. Because the destinations of the probes are random, for a specific queue q and a specific probe p, the probability that p passes through q is

$$\frac{1}{N}$$

Since there are $2NK$ probes in total, the probability that q is not hit by any of the probes is

$$(1-\frac{1}{N})^{2NK}$$

So the probability that there exists a level-1 queue, such that it is not hit by any probes can be bounded by

$$\begin{aligned}\mathbb{P}(\exists\ q \text{ not hit by any probe}) &\leq \sum_{i=1}^{N}\mathbb{P}(q_i \text{ not hit by any probe})\\ &= N(1-\frac{1}{N})^{2NK} \approx Ne^{-2K}\end{aligned}$$

For level-2 queues, namely the ToR uplink and L2 downlink queues, each has number N_2. Conditioning on a probe reaching level 2, which queues it hits is uniformly random. So for a level-2 queue q and a probe p, we have

$$\mathbb{P}(q \text{ hit by } p | p \text{ traverse to level 2}) = \frac{1}{N_2}$$

Probe p goes to level 2 and above only if it is probing a server outside its own rack; this event has chance $\frac{N-R}{N}$ where R is the rack size. Combining with the previous equation we have

$$\begin{aligned}&\mathbb{P}(q \text{ not hit by } p)\\ =&\mathbb{P}(q \text{ not hit by } p | p \text{ traverse to level 2})\mathbb{P}(p \text{ traverse to level 2})\\ &+\mathbb{P}(q \text{ not hit by } p | p \text{ not traverse to level 2})\mathbb{P}(p \text{ not traverse to level 2})\\ =&(1-\frac{1}{N_2})(1-\frac{R}{N})+1\cdot\frac{R}{N}\\ =&1-\frac{1}{N_2}(1-\frac{R}{N})\end{aligned}$$

So

$$\begin{aligned}\mathbb{P}(\exists\, q \text{ not hit by any probe}) &\le \sum_{i=1}^{N_2} \mathbb{P}(q_i \text{ not hit by any probe}) \\ &= N_2 \left(1 - \frac{1}{N_2}(1 - \frac{R}{N})\right)^{2NK} \\ &= N_2 e^{-2\frac{N-R}{N_2}K}\end{aligned}$$

This argument can be repeated for any level j. Assume the size of a level-j block is R_j, and the queues at level-j is N_j.

$$\begin{aligned}\mathbb{P}(q \text{ not hit by any probe}) &\le \sum_{i=1}^{N_j} \mathbb{P}(q_i \text{ not hit by any probe}) \\ &= N_j \left(1 - \frac{1}{N_j}(1 - \frac{R_j}{N})\right)^{2NK} \\ &= N_j e^{-2\frac{N-R_j}{N_j}K}\end{aligned}$$

Remarks:

- All the parameters N_i, R_i are properties of the network topology. In other words, the probability bounds are customized for the specific topology.
- The bounds are linear with respect to N_i, the number of queues, but exponential in K, the number of random probes per server. As an example, with the topology fixed, if $K_{new} = K + 2.3$, then the probability of not covered will decrease by 10 folds.
- Suppose there is no under-subscription (so $N_i \le N$ for all i) and R_i is negligible compared with N. Then for any level, the probability of some queue not covered in that level is bounded by Ne^{-2K}. For a typical network with 10k servers with $K = 10$, this probability is roughly 2×10^{-5}. When there are l network layers, we have $2l - 1$ levels of queues. Applying a union bound across all levels give the desired upper bound on the coverage probability $1 - (2l - 1)Ne^{-2K}$.
- Over-subscription is good from a coverage-point of view, as there are fewer higher level paths to choose from. We note in reality, most networks are over-subscribed. At the same time, link speed sometimes are higher for upper layer switching. Both contribute to the fact that there are usually fewer links in the uppers layers.