Router Support for Fine-Grained Latency Measurements

Ramana Rao Kompella, *Member, IEEE, ACM*, Kirill Levchenko, Alex C. Snoeren, *Member, IEEE*, and George Varghese, *Member, IEEE, Fellow, ACM*

***Abstract*—An increasing number of datacenter network applications, including automated trading and high-performance computing, have stringent end-to-end latency requirements where even microsecond variations may be intolerable. The resulting fine-grained measurement demands cannot be met effectively by existing technologies, such as SNMP, NetFlow, or active probing. We propose instrumenting routers with a hash-based primitive that we call a Lossy Difference Aggregator (LDA) to measure latencies down to tens of microseconds even in the presence of packet loss. Because LDA does *not* modify or encapsulate the packet, it can be deployed incrementally without changes along the forwarding path. When compared to Poisson-spaced active probing with similar overheads, our LDA mechanism delivers orders of magnitude smaller relative error; active probing requires 50–60 times as much bandwidth to deliver similar levels of accuracy. Although ubiquitous deployment is ultimately desired, it may be hard to achieve in the shorter term; we discuss a partial deployment architecture called mPlane using LDAs for intrarouter measurements and localized segment measurements for interrouter measurements.**

***Index Terms*—Communication technology, computer networks, coordinated streaming, latency measurement, router.**

I. Introduction

AN INCREASING number of datacenter-based applications require end-to-end latencies on the order of milliseconds or even microseconds. Moreover, many of them further demand that latency remain stable, i.e., low jitter, for optimal performance. These applications range from storage-area networks (SANs) to interactive Web services that depend on large numbers of back-end services to niche—but commercially important—markets like automated trading and high-performance computing. Currently, most of these latency-sensitive applications are deployed on specialized and often boutique hardware technologies like InfiniBand and FibreChannel. Technology advances and market pressures, however, are leading to increasing use of hybrid technologies like FibreChannel over Etherent (FCoE) [16] that have the potential for increased performance crosstalk and complex partial failure modes. Vendors and operators are under increasing pressure to be able to provision and manage converged datacenter networks that meet these stringent specifications. Unfortunately, most of the currently available tools are unable to accurately measure latencies of these magnitudes, nor can they detect or localize transient delay variations or loss spikes. Hence, we propose a new mechanism to measure latency and loss at extremely small timescales, even tens of microseconds.

As a motivating example, consider a trading network that connects a stock exchange to a number of data centers where automatic trading applications run. In order to prevent unfair arbitrage opportunities, network operations personnel must ensure that the latencies between the exchange and each data center are within 100 μs of each other [36]. (A recent *InformationWeek* article claims that "a one-millisecond advantage in trading applications can be worth $100 million a year to a major brokerage firm" [24].)

Current routers typically support two distinct accounting mechanisms: SNMP and NetFlow. Neither are up to the task. SNMP provides only cumulative counters that, while useful to estimate load, cannot provide latency estimates. NetFlow, on the other hand, samples and timestamps a subset of all received packets; calculating latency requires coordinating samples at multiple routers (e.g., trajectory sampling [8]). Even if such coordination is possible, consistent samples and their timestamps have to be communicated to a measurement processor that subtracts the sent timestamp from the receive timestamp of each successfully delivered packet in order to estimate the average, a procedure with fundamentally high space complexity. Moreover, computing accurate time averages requires a high sampling rate, and detecting short-term deviations from the mean requires even more. Unfortunately, high NetFlow sampling rates significantly impact routers' forwarding performance and are frequently incompatible with operational throughput demands.

Thus, operators of latency-critical networks are forced to use external monitoring mechanisms in order to collect a sufficient number of samples to compute accurate estimates. The simplest technique is to send end-to-end probes across the network [23], [30], [33]. Latency estimates computed in this fashion, however, can be grossly inaccurate in practice. In a recent Cisco study, periodic probes sent at 1-s intervals computed an average latency of under 5 ms, while the actual latencies as reported by a hardware monitor were around 20 ms with some bursts as high as 50 ms [29, Fig. 6]. Capturing these effects in real networks requires injecting a prohibitively high

Manuscript received September 20, 2010; revised May 07, 2011; accepted August 23, 2011; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Papagiannaki. Date of publication March 23, 2012; date of current version June 12, 2012. This work was supported in part by the National Science Foundation under Award CNS-0831647 and Cisco Systems. Portions of this manuscript appeared in the ACM SIGCOMM, Barcelona, Spain, August 17–21, 2009, and the ACM Re-Architecting the Internet (ReArch), Rome, Italy, December 1, 2009.

R. R. Kompella is with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (e-mail: kompella@cs.purdue.edu).

K. Levchenko, A. C. Snoeren, and George Varghese are with the Department of Computer Science, University of California, San Diego, La Jolla, CA 92093 USA (e-mail: klevchen@cs.ucsd.edu; snoeren@cs.ucsd.edu; varghese@cs.ucsd.edu).

Digital Object Identifier 10.1109/TNET.2012.2188905

1063-6692/$31.00 © 2012 IEEE

rate of probe packets. For these reasons, operators often employ external passive hardware monitors (e.g., those manufactured by Corvil [2]) at key points in their network. Unfortunately, placing hardware monitors between every pair of input and output ports is cost-prohibitive in many instances.

Instead, we propose the Lossy Difference Aggregator (LDA), a low-overhead mechanism for fine-grain latency and loss measurement that can be cheaply incorporated within routers to achieve the same effect. LDA has the following features.

- *Fine-granularity measurement:* LDA accurately measures loss and delay over short timescales while providing strong bounds on its estimates, enabling operators to detect short-term deviations from long-term means within arbitrary confidence levels. Active probing requires 50–60 times as much bandwidth to deliver similar levels of accuracy, as demonstrated in Section IV-C.
- *Low overhead:* Our suggested 40-Gb/s LDA implementation uses less than 1% of a standard networking ASIC and 72 kb of control traffic per second, as detailed in Section V.
- *Customizability:* Operators can use a classifier to configure an LDA to measure the delay of particular traffic classes to differing levels of precision, independent of others, as discussed in Section V.

While researchers are often hesitant to propose new router primitives for measurement because of the need to convince major router vendors to implement them, we observe several recent trends. First, router vendors are already under strong financial pressure from trading and high-performance computing customers to find low-latency measurement primitives. Second, the advent of merchant silicon such as Broadcom and Marvell has forced router vendors to seek new features that will avoid commoditization and preserve profit margins. Hence, we suggest that improved measurement infrastructure might be an attractive value proposition for legacy vendors.

LDA forms the key building block of a network-wide architecture we propose for collecting fine-grained latency measurements called mPLANE. By using LDA for intrarouter measurements and active probes for link measurements, mPLANE enables fine-grained latency measurements across the entire network for network operators to localize any end-to-end latency spikes. We also propose an incremental deployment strategy so that parts of the network can be upgraded in a more gradual fashion. without requiring all routers to be upgraded at the same time.

The remainder of this paper is organized as follows. We begin in Section II by discussing the mPLANE architecture. Section III introduces the Lossy Difference Aggregator and provides analytical bounds on its performance. We evaluate LDA through simulation in Section IV and propose a potential hardware realization in Section V.

II. mPLANE ARCHITECTURE

In this section, we discuss the architecture of mPLANE that provides router support for fine-grain latency measurements. Before we describe the architecture, we first discuss the latency measurement requirements in different domains.

A. Requirements

An application's latency requirements depend greatly on its intended deployment scenario. In the datacenter environment, back-end storage-area networks are among the most demanding applications, and FiberChannel has emerged to deliver similar latencies between CPUs and remote disks, replacing the traditional I/O bus. Automated trading applications have even more stringent requirements, as delays larger than 100 μs can lead to arbitrage opportunities that can be leveraged to produce large financial gains. Additionally, high-performance computing applications have also begun to place increased demands on datacenter networks. Infiniband, the *de facto* interconnect, offers latencies of 1 μs or less across an individual switch and 10 μs end to end. While obsessing over a few microseconds may seem excessive to an Internet user, modern CPUs can "waste" thousands of instructions waiting for a response delayed by a microsecond.

While latency demands obviously cannot be as stringent in the wide area, a number of applications are quite sensitive to delay on the order of tens to hundreds of milliseconds. For instance, interactive multimedia games (e.g., *World of Warcraft*, *Quake*) require fast-paced interaction and can be severely degraded by Internet latencies. While techniques such as dead-reckoning can ameliorate the impacts, latencies of more than 200 ms are considered unplayable [5]. Interactive applications such as video conferencing have strict limits on buffer length, and excess jitter substantially diminishes the user experience. For example, Cisco's recommendations for VoIP and video conferencing include an end-to-end, one-way delay of no more than 150 ms, jitter of no more than 30 ms, and less than 1% loss [35]. While we focus initially on the datacenter environment—where the need is more immediate—we discuss later how the mPLANE architecture can be applied to the wide area as well.

Metrics: Each of these domains clearly needs the ability to measure the average latency and loss on paths, links, or even link segments. However, in addition, the standard deviation of delay is important because it not only provides an indication of jitter, but further allows the calculation of confidence bounds on individual packet delays. For example, one might wish to ensure that, say, 98% of packets do not exceed a specified delay. (The maximum per-packet delay would be even better, but we show below that it is impossible to calculate efficiently.)

B. Key Idea: Segmented Measurement

The majority of operators today employ active measurement techniques that inject synthetic probe traffic into their network to measure loss and latency on an end-to-end basis [23], [30], [33]. While these tools are based on sound statistical foundations, active measurement approaches are inherently intrusive and can incur substantial bandwidth overhead when tuned to collect accurate fine-grained measurements, as we demonstrate later.

Rather than conduct end-to-end measurements and then attempt to use tomography or inference techniques [3], [6], [10], [17], [26], [37], [38] to isolate the latency of individual segments [18], [39], we propose to instrument each segment of the network with our new measurement primitive. Thus, in our model, every end-to-end path can be broken up into what we call *measurement segments*. For example, as shown in Fig. 1, a

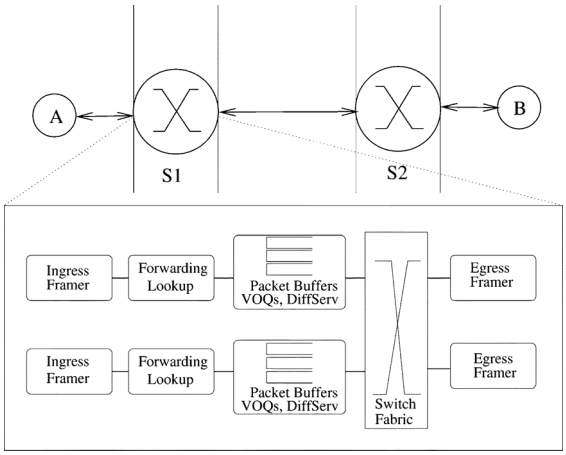


Fig. 1. Path decomposed into measurement segments.

path from endpoint A to endpoint B via two switches, $S1$ and $S2$, can be decomposed into five measurement segments: a segment between A and the input port of $S1$, a segment between the ingress port of $S1$ and the egress port of $S1$, a segment between the egress port of $S1$ and the ingress port of $S2$, a segment between the ingress port of $S2$ and the egress port of $S2$, and a final segment between the egress port of $S2$ and B.

A typical measurement segment extending from just after reception on a router's input port to just before transmission on the output side has the potential for significant queuing. However, deployments concerned with low latency (e.g., less than 100 μs) necessarily operate at light loads to reduce queuing delays, and thus, latencies are on the order of tens of microseconds. Such a router segment can be further decomposed as shown in the bottom of Fig. 1 into several segments corresponding to internal paths between key chips in the router (e.g., forwarding engine to the queue manager, queue manager to the switch fabric). Such decomposition allows the delay to be localized with even finer granularity within a router if queuing occurs and may facilitate performance debugging within a router.

C. Architecture

The MPLANE architecture utilizes the segmented measurement idea described before. Each router consists of two main components—external and internal measurement modules. The external measurement module is responsible for measuring link-level properties from the egress of a router to the ingress of the adjacent one. The internal measurement module measures performance metrics of interest (average delay, variance, and loss) for all the internal forwarding paths within a router (e.g., every ingress–egress pair). The router reports the measurements generated by these modules to a centralized monitoring station that is equipped with a topology monitor (e.g., OSPF monitor [32]). The monitoring station first identifies the forwarding paths in the network using the topology monitor, and then isolates the root causes of any perceived problems by identifying the relevant router- and link-level measurements responsible for that path and observing their properties.

External Measurements: Direct link measurements can be conducted using lower frequency probes since there is usually not much variability in link metrics. One way to essentially remove the need for any additional probes is to rely on packets that are already exchanged between routers (e.g., time synchronization packets, or OSPF Hello probes as done in [31]). However, this scheme is likely not going to work when measuring the properties of virtual segments between two upgraded routers that are not directly adjacent to each other. For such measurements, MPLANE requires that routers inject direct active probes to another upgraded router. (We will come back to the issue in Section VI when we explain how MPLANE can be incrementally deployed.)

Internal Measurements: In many real routers, forwarding metrics (e.g., loss, delay) depend on the forwarding class more than the particular flow. For example, all flows traveling between the same input and output ports of a router in a given QoS class are often treated identically in terms of queuing and switch scheduling. Thus, we group such flows into what we call a measurement equivalence class (MEC). We propose the use of a scalable measurement primitive that can report latency measurements per MEC within the router. In Section III, we will discuss a new scalable primitive called LDA that can achieve this in a scalable fashion. (Our architecture, however, is not tied to LDA; any scalable measurement primitive that can report our measurements would work equally well.)

For n routers in the network, the number of measurements in the network equals $\sum d_i^2$, where d_i is the degree of router i, assuming we instrument an LDA-like primitive to report latencies on a per-ingress–egress interface pair in each router. LDAs, as we shall see in Section III, are passive data structures that do not inject any active probes. Thus, a positive side-effect of LDA-like primitives is that there is very minimal probe bandwidth, which in turn allows our architecture to scale quite well, in addition to allowing direct fault isolation of end-to-end problems.

III. Segment Measurement Using LDA

We focus on a single *measurement segment* between a sender and a receiver. We assume that the segment provides first-in–first-out (FIFO) packet delivery. While the sender and receiver could be, in general, arbitrary measurement points, it is difficult to guarantee FIFO packet delivery across two routers. Thus, in this paper, we focus on segments such as an ingress–egress interface pair of a router where packet ordering is typically guaranteed. In practice, packets are commonly load-balanced across router interfaces, but since TCP interacts poorly with reordering, packets are typically resequenced before sending out on the egress interface. In such a case, we assume that measurement is conducted after the resequencing points so that the FIFO assumption is still valid.

We further assume that the segment endpoints are tightly time synchronized (to within a few microseconds). Microsecond synchronization is easily maintained within a router today and exists within a number of newer commercial routers. These routers use separate hardware buses for time synchronization that directly connect the various synchronization points within a router such as the input and output ports; these buses bypass the packet paths that have variable delays. Hence, the time interval between sending and receiving of synchronization signals is small and fixed. Given that most of the variable delays and loss is within routers, our mechanism can immediately be deployed within routers to allow diagnosis of the majority of latency

problems. Microsecond synchronization is also possible across single links using proposed standards such as IEEE 1588 [15]. Router vendors such as Cisco have already begun to incorporate this standard into their next-generation switches [1]. If the clocks at sender and receiver differ by e, then all latency estimates will have an additive error of e as well.

We divide time into measurement intervals of length T over which the network operator wishes to compute aggregates. We envisage values of T on the order of a few hundred milliseconds or even seconds. Smaller values of T would not only take up network bandwidth, but would generate extra interrupt overhead for any software processing control packets. For simplicity, we assume that the measurement mechanism sends a single (logical) control packet every interval. (In practice, it may need to be sent as multiple frames due to MTU issues.)

Thus in our model, the sender starts a measurement interval at some absolute time t_s by sending a Start control message. The sender also begins to compute a synopsis S on all packets sent between t_s and $t_s + T$. At time $t_s + T$, the sender also sends an End control message. If the receiver gets the Start control message (since control messages follow the same paths as data messages, they can be lost and take variable delay), the receiver starts the measurement process when it receives the Start control message. The receiver computes a corresponding synopsis R on all packets received between the Start and End control messages. The sender sends synopsis S to the receiver in the End control message so that it can compute latency and loss estimates as a function of S and R.

Note that the receiver can start much later (depending on the actual one-way latency between the sender and receiver) than the sender, but the goal is merely that the sender and receiver compute the synopses over the same set of packets. This is easily achieved if the link is FIFO and the Start and End control messages are not lost. Loss of control packets can be detected by adding sequence numbers to them. If either the Start or End control packets are lost, the latency estimate for an interval is unusable. Note that this is no different from losing a latency estimate if an active probe is lost.

We assume that individual packets do not carry link-level timestamps. If they could, trivial solutions are possible where the sender adds a timestamp to each packet, and the receiver subtracts this field from the time of receipt and accumulates the average and variance using just two counters. Clearly, IP packets do not carry timestamps across links where the TCP timestamp option is end-to-end. While timestamps could be added or modified within a switch, adding a 32-bit timestamp to every packet can add up to 10% overhead to the switch-fabric bandwidth. Adding an extra header requires intrusive changes to all components along the router forwarding path, including third-party components such as TCAMs, making it hard in practice. Furthermore, loss would still need to be computed with state accumulated at both ends. We will show that by adding only a modest amount of state beyond that required for loss measurements, we can also provide fine-grain measurements of the average and standard deviation of latency.

A. Coordinated Streaming

We measure the goodness of a measurement scheme by its accuracy for each metric (in terms of relative error), its storage overhead, bandwidth requirements, and its computational overhead. A naive solution to the measurement problem is for the sender to store a hash and timestamp of each sent packet and for the receiver to do the same for each received packet. At the end of the interval, the sender sends the hashes and timestamps for all N packets to the receiver, who then matches the send and receive timestamps of successfully received packets using the packet hashes and computes the average. Indeed, Papagiannaki *et al.* used a similar approach in their study of router delays [27]. Unfortunately, the naive solution is very expensive in terms of our performance measures as it takes $O(N)$ state at the sender and $O(N)$ bandwidth to communicate the timestamps. N can be large. For example, if measurement interval is 1 s, and the segment operates at 40 Gb/s, then N can be as large as 125 million 40-B packets. We aim for a scheme that is well within the capabilities of today's ASICs.

The quest for efficient solutions suggests considering streaming algorithms. Several streaming algorithms are already popular in the networking community for various applications such as finding heavy hitters [11], counting flows [12], estimating entropy [20], and computing flow-size distributions [19], [21]. The standard setting for streaming problems considers a single computational entity that receives a stream of data: The goal is to compute a function f of a single set of N values using a synopsis data structure that is much smaller than N.

Latency measurement, by contrast, is what we term a *coordinated streaming* problem with loss. In the general setting, we have two computational entities A and B. There are two streams of data values a_x and b_x; a_x is the time packet x left A, and b_x is the time it is received at B. Some packets are lost, so b_x may be undefined. The goal here is to compute some function f of the set of (a_x, b_x) pairs. For measuring average latency, the function is $\sum_x (b_x - a_x)$ over the cardinality of the set of packets for which a_x is defined (i.e., packets that are received and not lost). For measuring variance, the function is $\sum_x (b_x - a_x)^2$ over the received packets. For measuring, say, the maximum delay, the function would be $\max(b_x - a_x)$. In all cases, the function requires a pairwise matching between a received data item and the corresponding sent item—a requirement absent in the standard streaming setting.

The coordinated streaming setting is strictly harder than the standard setting. To see this, observe that computing the maximum data item in the stream is trivial in a standard streaming using $O(1)$ space and $O(1)$ processing. However computing the maximum delay requires $\Omega(N)$ space, even without the assumption of loss. (The proof is a straightforward reduction from Set Disjointness as in Alon *et al.* [4].) Despite this negative result for the maximum delay, we will show that approximating both average and standard deviation of delay can be done efficiently. Next, we describe the LDA, a mechanism that estimates these statistics.

B. LDA

A Lossy Difference Aggregator is a measurement data structure that supports efficiently measuring the average delay and standard deviation of delay. Both sender and receiver maintain

an LDA; at the end of a measurement period—in our experiments, we consider 1 s—the sender sends its LDA to the receiver and the receiver computes the desired statistics. The only additional requirements are tight time synchronization between sender and receiver, a requirement shared by all one-way delay measurement mechanisms, and consistent packet ordering (i.e., no packet reordering) at the sender and receiver.

To better explain the LDA, we begin with the simplest average delay measurement primitive—a pair of counters—and then develop the full LDA.

1) No Loss: To start, consider the problem of (passively) measuring the average latency between a sender A and a receiver B. A natural approach is a pair of timestamp accumulators, adding up packet timestamps on the sender and receiver sides, and a packet counter. The average delay is then just the difference in timestamp accumulators between sender and receiver, divided by the number of packets: $(T_B - T_A)/N$. Of course, if packets are lost, this approach fails: The sender's timestamp accumulator T_A will include the timestamps of the lost packets, while the receiver's will not.

2) Low Loss: Consider the case of exactly one loss. If we randomly split the traffic into m separate "streams" and compute the average latency for each such "stream" separately, then a single loss will only make one of our measurements unusable; we can still estimate the overall average latency using the remaining measurements.

Practically speaking, we maintain an array of several timestamp accumulators and packet counters (collectively called a *bank*). Each packet is hashed to a *row index* in the range 1 to m, and the corresponding timestamp accumulator and packet counter are updated as before. By using the same row hash function on the sender and receiver, we can determine exactly how many packets hashed to each accumulator–counter pair as well as how many of them were lost. Note that the sum of the receiver's packet counters gives us the number of packets received and the sum of the sender's packet counters, the number of packets sent; the difference gives the number of lost packets.

If a packet is lost, the value of the sender's packet counter at the row index to which the packet was hashed will be one greater than the value of corresponding packet counter on the receiver. We call such a row index *unusable* and do not use it in calculating our average delay estimate. The remaining usable row indices give us the average delay for a *subset* of the packets. With a single loss, m accumulator–counter pairs are roughly equivalent to sampling roughly every $m-1$ in m packets, providing a very accurate estimate of the overall average latency. The number of packets that hashed to a usable row index is the *effective sample size* of the latency estimate. In other words, it is as if we had sampled that many packets to arrive at the estimate. In general, for a small number of losses L, the expected effective sample size is at least a $(1 - L/m)$ fraction of the received packets.

Example: Fig. 2 shows an example configuration with $m = 4$ and exactly one lost packet that hashed to the second accumulator–counter pair. The sum of packet delays from the other three usable accumulators is $(180-120)+(37-15)+(14-6) = 90$; the effective sample size is $5+2+1 = 8$. The estimated delay is thus $90/8 = 11.25$.

3) Known Loss Rate: For larger loss rates, we need to sample the incoming packets to reduce the number of potentially unusable rows. Sampling can easily be done in a coordinated fashion at receiver and sender by (once again) hashing the packet contents to compute a sampling probability. Thus, we ensure that a packet is sampled at the receiver only if it is sampled at the sender. At sample rate p, we expect the number of lost packets that are recorded by the LDA to be pL, so that the expected number of usable rows is at least $m - pL$. Of course, packet sampling also reduces the *overall* number of packets counted by the LDA, reducing the accuracy of the latency estimate. In Section III-D, we will address this issue formally. Intuitively, however, we can see that for p on the order of m/L, we can expect at least a constant fraction of the accumulator–counter pairs to suffer no loss and therefore be usable in the latency estimator.

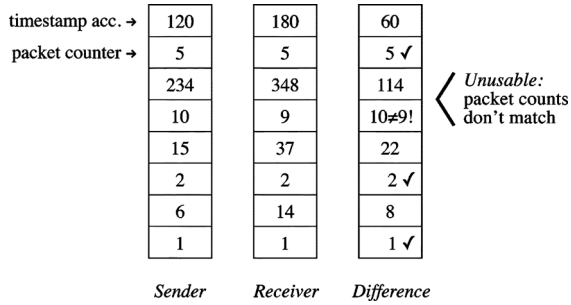


Fig. 2. Computing LDA average delay with one bank of four timestamp accumulator–counter pairs. Three pairs are usable (with 5, 2, and 1 packets), while the second is not due to a packet loss. Thus, the average delay is $(60 + 22 + 8)/(5 + 2 + 1)$.

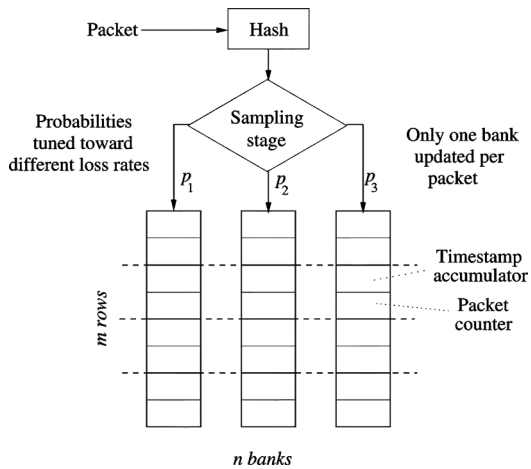


Fig. 3. LDA with n banks of m rows.

4) Arbitrary Loss Rate: So far, we have seen that a single bank of timestamp accumulators and packet counters can be used to measure the average latency when the loss rate is known *a priori*. In practice, of course, this is not the case. To handle a range of loss rates, we can use multiple LDA banks, each tuned to a different loss rate (Fig. 3). In our experiments, we found that two banks (tuned toward the lowest and highest loss rates) is a reasonable choice for loss rates under 5%.

At first glance, maintaining multiple banks seems to require maintaining each bank independently and then choosing the best bank at the end of the measurement period for computing the estimate. However, we can structure a multibank LDA so that only one bank needs to be updated per sampled packet.

The trick is to have *disjoint* sample sets so that each packet is sampled by a single bank, if at all. This way, only a single bank needs to be updated and later, during post-processing, no packet is double-counted. Furthermore, as a practical matter, a single row hash function can be shared by all banks. Each packet is hashed to a row uniformly and to a bank *nonuniformly* according to bank sampling probabilities $p_1, p_2, \ldots, p_n$. For nonuniform sampling probabilities that are powers of 1/2, this can be implemented by hashing each packet to an integer *uniformly* and using the number of leading zeros to determine the one bank that needs to be updated. We can compute the average delay by combining all usable elements across all banks. The full $m \times n$ LDA is shown in Fig. 3. For example, consider two banks having sampling probabilities $p_1 = 1/2^3$ and $p_2 = 1/2^7$. Each packet is hashed to an integer. If the first seven bits are zero, then bank 2 is updated. Otherwise, if the first three bits are zero, then bank 1 is updated. Otherwise, if the first three bits are not all zero, the packet is not sampled.

C. Update Procedure

Formally, the update procedure is as follows. Let x denote a packet, $h(x)$ the row hash function, and $g(x)$ the bank sampling hash function. The row hash function $h(x)$ maps x to a row index distributed uniformly between 1 and m. The sampling hash function $g(x)$ maps x to bank j, where $g(x) = j$ with probability p_j. In our analysis, we assume that h and g are 4-universal (which is amenable to efficient implementation), although in practice this may not be necessary. We use the special value $g(x) = 0$ to denote that the packet is not sampled. Upon processing a packet x at time τ, timestamp τ is added to the timestamp accumulator at position $(h(x), g(x))$, and the corresponding packet counter is incremented. If $g(x) = 0$, then the packet is simply ignored. Using T to denote the $m \times n$ array of timestamp accumulators and S to denote corresponding array packet counters, the procedure is as follows:

1. $i \leftarrow h(x)$
2. $j \leftarrow g(x)$
3. **if** $j \geq 0$ **then**
4. $\quad T[i,j] \leftarrow T[i,j] + \tau$
5. $\quad S[i,j] \leftarrow S[i,j] + 1$
6. **end if**

D. Average Latency Estimator

From the discussion above, estimating the average latency is straightforward: For each accumulator–counter pair, we check if the packet counters on the sender and receiver agree. If they do, we subtract the sender's timestamp accumulator from the receiver's. If they do not, this accumulator–counter pair is considered *unusable*. The average delay is then estimated by the sum of these differences divided by the number of packets counted.

Formally, let $T_A[\cdot,\cdot]$ and $T_B[\cdot,\cdot]$ denote the $m \times n$ timestamp accumulator arrays of the sender and receiver, respectively, and $S_A[\cdot,\cdot]$ and $S_B[\cdot,\cdot]$ the corresponding packet counters. Call a position (i,j) *usable* if $S_A[i,j] = S_B[i,j]$. Let u_{ij} be an indicator for this event, that is, $u_{ij} = 1$ if (i,j) is usable, and $u_{ij} = 0$ otherwise. Define

$$T_A = \sum_{i=1}^{m} u_{ij} T_A[i,j] \quad T_B = \sum_{i=1}^{m} u_{ij} T_B[i,j].$$

T_A and T_B are the sum of the of the usable timestamp accumulators on the sender and receiver, respectively. By definition $u_{ij} S_A[i,j] = u_{ij} S_B[i,j]$, so let

$$S = \sum_{i=1}^{m} u_{ij} S_A[i,j] = \sum_{i=1}^{m} u_{ij} S_B[i,j].$$

The estimate, then, is

$$D = \frac{1}{S}(T_B - T_A).$$

The quantity S is is the *effective sample size* from which the average latency is calculated. In other words, if one were to sample and *store* packet timestamps, the number of packets sampled would need to be at least S to achieve the same statistical accuracy as the LDA. Using a Hoeffding inequality [13], it can be shown that

$$\Pr[|D - \mu| \geq \epsilon\mu] \leq 2e^{-\epsilon^2 S\mu^2/2\sigma^2} \tag{1}$$

where μ and σ are the actual mean and standard deviation of the delays. When $\sigma \approx \mu$ the estimate is very accurate given a reasonable effective sample size. Let R and L be the number of received and lost packets, respectively, so that $R + L = N$. For a single bank with design number of losses $L \geq m$, setting the packet sampling probability $p = \alpha m/(L+1)$, where α is a parameter to be optimized}, gives an expected effective sample size of

$$\mathrm{E}[S] \geq \alpha(1-\alpha) \cdot \frac{m}{L+1} \cdot R. \tag{2}$$

Note that if we were to *store* the sampled packets, the expected sample size would be just pR with a tight concentration around this value. However, because we are *not* storing the packets but recording them in the LDA, we pay a constant factor $(1-\alpha)$ penalty in the effective sample size and a higher variance. To maximize the bound, we set $\alpha = 0.5$ (in our evaluation).

E. Latency Standard Deviation

Note that we exploited the fact that the sum of the differences of receive and send packet timestamps is the same as the difference of their sum. While this reshuffling works for the sum, it does not work for the sum of squares. Despite this obstacle, we now show that the LDA can also be used to estimate the standard deviation of the packet delays. This is crucial because an accurate measure for standard deviation allows a network manager to compute tight confidence intervals on the delay, a highly desirable feature in a trading or high-performance computing applications.

Again, let us start by assuming no loss; we can correct for loss later using the same hashing technique as we used for the average. Consider the two timestamp sums we already keep at

the sender and receiver, T_A and T_B. If we take the difference, this is just the sum of packet delays. If we now square this difference, we get

$$\sum_x (b_x - a_x)^2 + \sum_{x \neq x'} (b_x - a_x)(b_{x'} - a_{x'}).$$

The first sum (of delays squared) is exactly what we need for computing the standard deviation since

$$\sigma^2 = \frac{\sum_x (b_x - a_x)^2 - \mu^2}{N} \tag{3}$$

but we also get unwanted cross terms. Fortunately, the cross terms can be eliminated using a technique introduced by Alon *et al.* [4]. The idea is to keep a slightly different timestamp accumulator on the sender and receiver: Instead of simply adding the timestamp, we add *or subtract* with equal probability based on a consistent hash. Using s_x to denote the ± 1 hash of the packet, we now have

$$\begin{aligned}
&\left(\sum_x s_x b_x - \sum_x s_x a_x\right)^2 \\
&= \left(\sum_x s_x (b_x - a_x)\right)^2 \\
&= \sum_{x,x'} s_x s_{x'} (b_x - a_x)(b_{x'} - a_{x'}) \\
&= \sum_x s_x^2 (b_x - a_x)^2 + \sum_{x \neq x'} s_x s_{x'} (b_x - a_x)(b_{x'} - a_{x'}).
\end{aligned} \tag{4}$$

Because of independence, the expectation of the cross terms $\mathrm{E}[s_x s_{x'}]$ is zero, giving us an unbiased estimator for the square of the delays squared.

So far, this implies that we keep a separate signed timestamp accumulator. Also, to deal with loss, we would have to keep an array of such counters, doubling the number of timestamp accumulators. Fortunately, we can mine the existing LDA. Observe that the sign hash s_x above can be computed using the low-order bit of the hash function we use to compute a row index in the full LDA. To achieve the same effect without adding additional memory, we use this low-order bit of the row hash value $h(x)$ as the sign bit, "collapsing" adjacent rows. (Thus, the estimator uses $\frac{1}{2}m$ rows.)

Define the collapsed $\frac{1}{2}m \times n$ timestamp accumulator and packet counter arrays as

$$\begin{aligned}
\tilde{T}_A(i,j) &= T_A[2i,j] - T_A[2i-1,j] \\
\tilde{T}_B(i,j) &= T_B[2i,j] - T_B[2i-1,j] \\
\tilde{S}_A(i,j) &= S_A[2i,j] + S_A[2i-1,j] \\
\tilde{S}_B(i,j) &= S_B[2i,j] + S_B[2i-1,j].
\end{aligned}$$

Let $\tilde{u}_{ij}$ be an indicator for a position being usable; that is, $\tilde{u}_{ij} = 1$ if $\tilde{S}_A(i,j) = \tilde{S}_B(i,j)$, and $\tilde{u}_{ij} = 0$ otherwise. As in the average latency estimator, let $\tilde{S} = \sum \tilde{u}_{ij} S_A(i,j)$. Our latency second frequency moment estimator is

$$F = \frac{1}{\tilde{S}} \sum_{i=1}^{m/2} \tilde{u}_{ij} (\tilde{T}_B(i, \tilde{u}_i) - \tilde{T}_A(i, \tilde{u}_i))^2. \tag{5}$$

Let $w_x = b_x - a_x$. It is straightforward to show that

$$\mathrm{E}[F] = \frac{1}{R} \sum_x w_x^2$$

as desired. We can then estimate the standard deviation of the delays using (3). The variance of F is upper-bounded by

$$\mathrm{Var}[F] \leq \frac{1}{R^2} \left(\frac{R - \tilde{S}}{\tilde{S}} \sum_x^{\mathrm{rec'd}} w_x^4 + 2 \sum_{x \neq x'}^{\mathrm{rec'd}} w_x^2 w_{x'}^2 \right).$$

For comparison, the basic estimator (4), which does not handle packet loss, corresponds to the case $R = \tilde{S}$ and has variance

$$\mathrm{Var}[F] \leq \frac{2}{R^2} \sum_{x \neq x'}^{\mathrm{rec'd}} w_x^2 w_{x'}^2.$$

By averaging several instances of the estimator as in [4], the variance can be reduced arbitrarily. In our experiments, however, we use the estimator (5) directly with satisfactory results. We note that this standard deviation estimate comes "for free" by mining the LDA data structure (designed for estimating average) for more information.

IV. Evaluation

Our evaluation has three major goals. First, we wish to empirically validate our analyses of an optimal LDA's estimates, both in terms of average delay and standard deviation. Second, we analyze various tuning options to select a set of practical configuration options. Finally, we use the resulting parameter settings to compare the efficacy of a practical LDA to the current cost-effective alternative: Poisson-modulated active probing. (We do not compare against special-purpose passive monitoring devices [36], as they are prohibitively expensive to deploy at scale.)

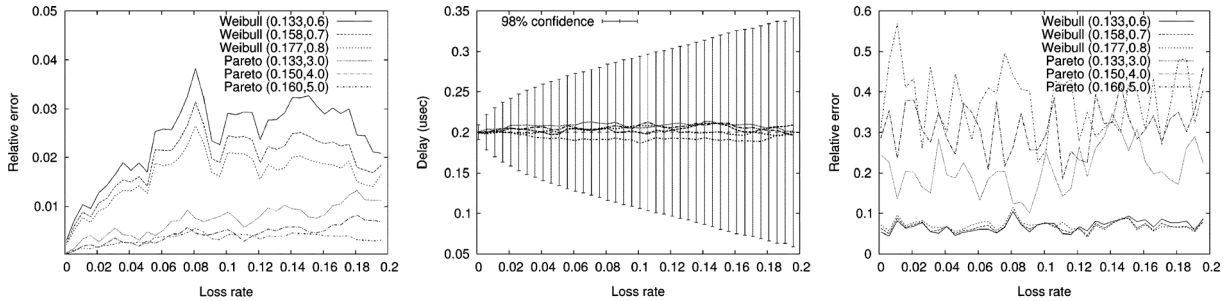
We have implemented a special-purpose simulator in C++ to facilitate our evaluation.[1] The simulator generates packet traces with various loss and delay distributions and implements several different variants of the LDA data structure, as well as active probing and the associated estimators needed to compare LDA to the active probing approach.

In an effort to evaluate LDA in realistic scenarios, we use delay and loss distributions drawn from the literature. In particular, Papagiannaki *et al.* report that packet delays recorded at a backbone router are well modeled by a Weibull distribution [27], with a cumulative distribution function

$$P(X \leq x) = 1 - e^{(-x/\alpha)^\beta}$$

with α and β representing the scale and shape, respectively. Unless otherwise noted, all of our experiments consider a Weibull distribution with their recommended shape parameter ($0.6 \leq \beta \leq 0.8$). For comparison purposes, we also simulated Pareto distribution generated according to the function $P(X \leq x) = 1 - (x/\alpha)^{-\beta}$ with α and β representing the scale and shape parameters, respectively, and β chosen between 3–5 so that the

[1]The main advantage of standard packages like ns2 is the library of preexisting protocol implementations like TCP, the vast majority of which are not needed in our experiments. Thus, we feel the remaining benefits are outweighed by the simplicity and significant speedup of a custom solution.



(a) (b) (c)

Fig. 4. Relative error and 98% confidence bounds of average delay estimates, and relative error of standard deviation estimate computed by LDA as a function of loss rate. Actual mean delay is 0.2 μs in all cases. In (b), each curve represents an LDA with different random seed on the same trace. (a) Relative error of average delay. (b) Estimated average delay. (c) Relative error of standard deviation.

delay values do not become too skewed and to ensure that the distributions have bounded variance.

For each packet, we essentially draw a delay value from the distribution and assign it to the packet. Therefore, while back-to-back packets may have completely uncorrelated delay values, the overall statistics such as mean delay and standard will however match the real packet delay distributions. Matching these statistics is sufficient for our purposes since LDA itself is oblivious to the exact sequence of delay values as it involves only addition of timestamps which is commutative. In order to ensure that sampled delay values do not cause packet reordering, we assign timestamps to packets such that two successive packets always differ by more than the delay of the first packet drawn from the distribution. In other words, we ensure that there is always only one packet in flight at any given instant by enforcing that a given packet begins transmission only after the previous packet has reached the receiver. This does not bias our results in any way since LDA does not care about the actual timestamps themselves; it is only the differences that matter.

LDA performance is independent of loss distribution within an interval, so most experiments use a uniform loss model for simplicity. For our comparisons to active probes—whose performance depends on the loss distribution—we use exponentially distributed loss episodes (as suggested by Misra *et al.* in their study of TCP behavior [25]), where each episode involves dropping a burst of packets (following the model of Sommers *et al.* [33]).

A. Validation

The main goal of the set of experiments described in this section is to empirically validate our analytical bounds using a simple single-bank LDA. In particular, we study the accuracy of LDA's estimates over different delay and loss distributions.

For these simulations, we configure the LDA to use $n = 1$ bank of $m = 1024$ counters. We simulate a 10-Gb/s OC-192 link that, assuming an average packet size of 250 B, carries roughly five million packets per second at capacity. (The choice of 250-B packets is arbitrary and results in round numbers; the functionality of LDA is not impacted by packet size.) We simulate a measurement interval of 1 s (so $N = 5\,000\,000$ and an average delay of 0.2 μs). For different distributions, we ensure consistency by adjusting the scale parameters appropriately to match the mean delay of 0.2 μs.

In order to isolate the effects of packet loss, for each experiment, we first generate a packet trace according to the desired delay distribution using a particular random seed, and then impose varying levels of loss. Each graph presented in this section uses the same random seed for the delay distribution.

We first verify empirically that the actual sample size obtained using our data structure matches expectation. For the purposes of this experiment, we assume that we know *a priori* the loss rate l; we compute the number of lost packets $L = N \cdot l$ and set the sampling probability accordingly as $p = \alpha m/(L+1)$, where $\alpha = 0.5$. Our findings indicate that our analytical bound is typically conservative; LDA captures more samples (almost twice as many) in simulation than the lower bound. (For brevity, we omit the actual graph.)

In Fig. 4(a), we plot the average relative error (defined as $|\text{true} - \text{estimated}|/|\text{true}|$) of LDA as we vary the loss rate. We obtain the ground truth by maintaining the full delay distribution. Each point corresponds to the average of the relative error across a set of 10 independent runs—i.e., the packet trace is the same, but the LDA selects a different random set of packets to sample during each run. The LDA is optimally configured for each loss rate as in the previous section. As expected, the relative error of the estimate increases as the loss rate increases because the number of available samples decreases with loss rate. While the curves all follow the same general trend, the estimates for the Weibull distributions are less accurate compared to Pareto. For the particular shape parameters we simulated, the Weibull distribution suffers from a larger variance than Pareto—variance is 0.123 at $\beta = 0.6$ for Weibull as compared to 0.013 at $\beta = 3$ for Pareto. LDA therefore requires more samples for Weibull to obtain the same accuracy level as Pareto. Even in the worst case of 20% loss, however, the estimates have less than 4% error on average. At low loss rates ($<0.1\%$), LDA estimates have less than 0.3% error. Results from similar experiments with a variety of random seeds are qualitatively similar; the relative error at loss rates of even 6% across different traces is never more than 3% with an average of about 0.2%.

Low error in expectation is nice, but some applications require guarantees of accuracy in every instance. To validate our error bounds, we focus on the delay distribution with the least accurate estimates from above, namely the $(\alpha = 0.133, \beta = 0.6)$ Weibull distribution. In Fig. 4(b), rather than report relative error, we graph the actual delay estimate computed by a representative five of the ten constituent runs in Fig. 4(a). In addition,

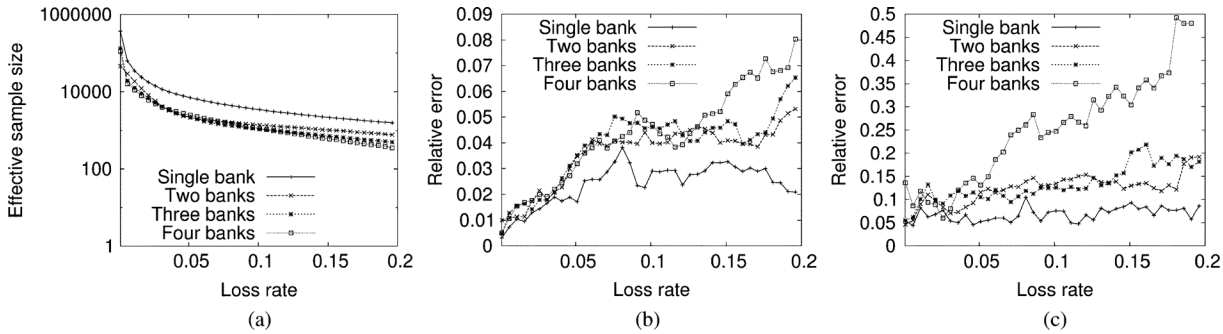


Fig. 5. Performance of various multibank LDA configurations. (a) Sample size. (b) Delay. (c) Standard deviation.

we plot the 98%-confidence bounds computed using (1). The actual confidence bound depends on the number of samples obtained by each LDA and, therefore, varies across instances. Each error bar shown in the figure corresponds to the most conservative bound computed based on the run that collected the smallest number of samples across the 10 runs from Fig. 4(a). While confidence decreases with higher loss rates, all of the individual estimates reported in our simulation remain quite close to the actual value. Results of other distributions are even tighter.

For the same setup as above, we also measure the accuracy of the LDA's standard-deviation estimator (obtained from the variance estimator). We plot the average relative error incurred for different distributions in Fig. 4(c). Estimates suffer from about 20%–50% relative error for Pareto to less than 10% error for Weibull distributions, independent of loss rate. The magnitude of the relative error obviously depends on the actual standard deviation of the underlying distribution, however. The true standard deviation of delay in the Weibull- and Pareto-distributed traces is about 0.35 and 0.11, respectively. Hence, the absolute error of LDA's standard-deviation estimator is similarly small in both cases. Delays in Internet routers are reported to be well modeled by a Weibull distribution [27], so relative error is likely to be small in practice.

B. Handling Unknown Loss Rates

Up to this point, we have configured each LDA optimally for the actual loss rate. Obviously, any real deployment will need to be configured for a range of loss rates. Here, we evaluate the efficacy of various configurations of multibank LDAs over a range of loss rates. In particular, each bank within the LDA is tuned ($p = \alpha m/(L+1), \alpha = 0.5$ as before) to a different target loss rate. We consider three alternatives, each with the same total number (1024) of counters: two banks of 512 counters tuned toward loss rates of 0.005 and 0.1; three banks with roughly one-third of the counters tuned toward loss rates of 0.001, 0.01, and 0.1; and, finally, four banks of 256 counters each tuned for loss rates of 0.001, 0.01, 0.05, and 0.1, respectively. These particular configurations are arbitrary; operators may find others better suited for their networks.

We present results along the same three dimensions considered previously—effective sample size, relative error of delay, and standard deviation estimates—in Fig. 5. To facilitate comparison, we continue with the same uniform loss and Weibull delay distributions and replot the optimal single-bank case configured for the actual loss rate as shown in Fig. 4(a) and (c).

Fig. 5(a) shows that while practical configurations collect fewer samples than optimal, the absolute value is not too far from our analytical estimates for the single-bank case. The delay and standard deviation curves in Fig. 5(a) and (b) follow a similar trend. The LDAS perform comparably across the ranges of loss, although the four-bank LDA performs the worst of the three when the loss rates are high. The number of buckets invested by the four-bank LDA tuned toward high loss rates (10%) is low, so it struggles to keep up. We note, however, that most real networks operate at low loss rates—typically substantially less than 5%. In conclusion, we expect a two-bank LDA configuration tuned to relatively low loss rates will be appropriate for most deployments.

C. Comparison to Active Probes

We compare the accuracy of the delay and standard deviation estimates obtained using the two-bank LDA to those that are obtained using Poisson-distributed active probes (such as those used by the well-known `zing` tool [23]) for various probe frequencies. Note that these experiments are only for comparing the accuracy of these two schemes; the implementational complexity of these schemes is quite different. LDA requires hardware modifications within the router, while implementing active probes to measure latency between router interface pairs requires additional measurement boxes that may be administratively difficult to provision.

The accuracy of the active probing approach depends critically upon the frequency, so we provide a selection of natural comparison points. One approach is to hold communication overhead constant. First, we consider an active-probing mechanism that communicates as frequently as LDA—once an interval, or 1 Hz. In practice, however, the LDA data structure is too large to fit into one MTU-sized packet—an Ethernet implementation would actually need to send seven separate packets per interval assuming 1024×72 bits $\approx$72 kb for the data structure and 1500-B packets. Thus, to be fair in terms of number of packets per second, we also use a probing frequency of 7 Hz. Moreover, probe packets are much smaller (containing only one timestamp and no counters), so holding bandwidth constant—as opposed to packet count—results in a probing rate of about 144 Hz (assuming probe packets of size 64 B). As we shall see, however, none of these rates approach the accuracy of LDA; hence, we also plot a frequency that delivers roughly equivalent performance: 10 000 Hz.

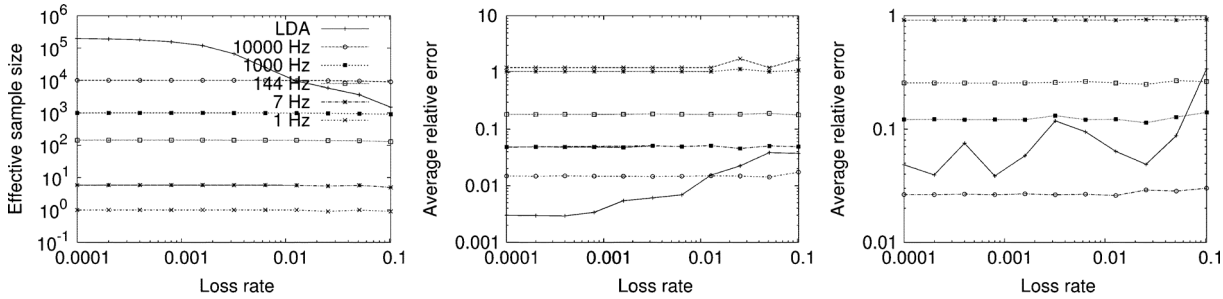


Fig. 6. (a) Sample size, (b) delay, and (c) standard deviation estimates obtained using a two-bank LDA in comparison with active probing at various frequencies. Log-scale axes.

We generate Poisson-modulated active probes by injecting probe packets at intervals distributed according to a Poisson process with the desired average interarrival time, and then subjecting the probe packets to the same delay distribution as the regular traffic. In a normal queue, adding an active probe affects the queuing dynamics—for example, it may cause packets behind to experience higher delays and in some cases, even be dropped. We do not, however, recreate such effects on packets behind active probes because packet delays are already based on a distribution, and simulating such affects will cause delays to deviate from the distribution. Thus, the delays of regular packets are not impacted by the presence of active probes; only their timestamps are shifted.

For these experiments, we continue to use the same Weibull delay distribution as before, but with exponentially distributed (as opposed to uniform) loss episodes with each episode consisting of about 100 packets. In Fig. 6, we compare the effective sample size, average relative error in the delay, and standard deviation estimators using active probes at various frequencies as well as LDA.

Fig. 6(a) clearly shows the impact of increased probe frequency: more samples. As before, each point represents the average of 10 runs. The number of samples collected by active probes decreases by a small amount as the loss rate increases due to the lost probes. While the number of effective samples obtained by LDA decreases more rapidly, the sample size remains far larger than those obtained by all but the most aggressive active probing rates under significant $(>1\%)$ loss. Consequently, the average relative error observed by LDA (0.2%–4%) is significantly lower than that for active probes with an equivalent number of packets (almost 100%) as shown in Fig. 6(b). Even when we hold the measurement bandwidth steady across LDA and active probes (144 Hz), we observe at least an order of magnitude (11% compared to less than 1% at loss rates less than 1%) difference in the relative error between the two. While the improvement in standard deviation estimates is not as stable as the average delay estimates, LDA is still considerably more accurate (3%–9% versus $\approx 15\%$) over realistic $(<5\%)$ loss rates. Overall, only the 10 000 Hz probing rate provides accuracy approaching LDA. Said another way, active probing requires 50–60 times as much bandwidth to achieve similar results.

Perhaps more importantly, however, LDA is significantly more reliable. In our experiments, we found that the 98%-confidence intervals for the constituent LDA runs from Fig. 6(a) are quite small—generally within 25% of the mean (for loss rates less than 0.1%) and less than a factor of two even at 10% loss. Empirically, however, each estimate is well inside the analytical envelope. Focusing on loss rate regimes that the LDA was tuned for, e.g., 0.16%, the maximum relative error across all runs of LDA was 0.17%. The same cannot be said for active probing, however, which had runs with relative errors as large as 87% for 7 Hz and 13% for 144 Hz. Once again, only 10 000-Hz probes were competitive, with a maximum relative error of 0.15%.

V. Hardware Realization

We have sketched out the base logic for LDA that we estimate takes less than 1% of a low-end $10 \times 10\,\text{mm}^2$ networking ASIC, using 400-MHz 65-nm process. The logic is flow-through, i.e., it can be inserted into the path of a link between the sender and receiver end without changing any other logic allowing easy incremental deployment. A minimal implementation would place a single LDA together with MAC logic at ingress and egress links.

We present a strawman design of LDA here. At the sender (receiver), the first X (say 50) bytes of the packet are sent to the logic, which then determines the control or data packet using an Ethernet type field. For data packets, an optional classifier can be used to filter specific types of packets. The update logic computes a hardware hash based on a few bytes of each packet. H3 hash functions [28], for example, can be implemented efficiently in hardware using XOR arrays and can be easily modified. Our estimates use a Rabin hash whose loop is unrolled to run at 40 Gbps using around 20 000 gates. The high-order bits of the 64-bit hash selects the sampling probability, which in turn determines which bank is selected. For example, if there are two banks, which are selected with probabilities 1/2 and 1/64, the six high-order bits are used. If a bank is selected, the low-order bits of the hash are used to post a read to the corresponding bank. For example, if each bank has 1024 counters, we use the 10 low-order bits. The update logic then reads the 72-bit value stored at the indicated location. The first 32 bits are a simple packet counter that is incremented. The last 40 bits are a timestamp sum (allows nanosecond precision) to which the current value of the hardware clock is added. The updated value is then written back to the same location.

The sender-side logic conceptually generates control packets at the end of each measurement interval. Control packets are sequence-numbered so that loss of control packets translates into a measurement interval being ignored. When the receiver logic receives the sender's control packets and updates its own, it sends the control packets to a line-card processor that computes delay, loss, and variance estimates in software that it can then report to a management station on demand. The simplest design of the control is to keep two copies of each counter so that it can work on reading and zeroing LDA counters for a prior interval into control packet(s) concurrently with the update process. Alternately, two control packets, one for the end of an interval and the other sent T' seconds later for the start of a new one, can be used. Keeping T' small will ensure only a small number of samples (say 100) are ignored.

The logic for counters is placed in SRAM, while the remaining logic is implemented in flops. In a 65-nm 400-MHz process, 1000 SRAM counters of 72 bits each take 0.13 mm^2. While the size for the hash logic is about 20 000 gates, we conservatively estimate another 30 000 gates for the classifier (a simple mask-and-compare to *one* specified header), header extraction, and counter update, yielding a total of around 50 000, or approximately 0.1 mm^2 in a 65-nm process. The grand total is around ≈ 0.23 mm^2. Even if we double the width of the counters and keep two copies of the entire data structure (to handle sender and receiver logic), an LDA still represents less than 1% of the area of the lowest-end (10×10 mm^2) ASICs on the market today.

VI. Incremental Deployment of mPlane

In this section, we discuss how mPlane can be incrementally deployed. In typical ISP networks, measurement servers are typically connected to the edge routers, and $O(n^2)$ active probes are injected between all pairs of servers to measure the health of the network. Tomographic techniques (e.g., [6] and [10]) can then be applied to infer individual hop and link latencies. In datacenter networks, it is not clear that this approach works well because: 1) active probes require way too much probing bandwidth for them to be accurate, as our experiments indicate in Section IV; 2) inference is typically underconstrained and may not be accurate. We can solve the second problem by just essentially placing measurement servers across each and every link in the network. We cannot, however, solve the first problem unless we upgrade routers with data structures such as LDAs.

In a clean-slate deployment, all routers are upgraded to include native latency measurement techniques such as LDAs; we refer to these upgraded routers as *m-routers*. In this scenario, there is no need for any measurement servers for fault localization purposes, although they may still be required for end-to-end measurements. Given it may be difficult to perform such a "forklift" upgrade, we propose the following incremental deployment strategy that blends the active probes approach (with reduced accuracy measurements in some portions of the network) with upgraded routers that support LDAs (with high-fidelity measurements in those routers).

Our incremental deployment stragey involves three main steps. In the first step, the set of measurement servers connected directly to the m-routers is removed since its functionality

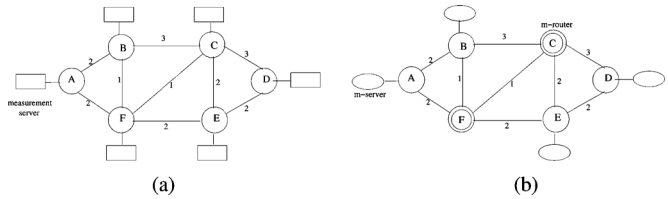


Fig. 7. Partial deployment and clean-slate design of the mPlane architecture. (a) Current topology. (b) Partially upgraded topology.

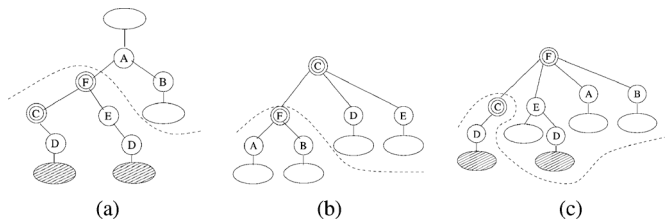


Fig. 8. Few shortest-path trees constructed locally by the m-servers and the m-routers to determine which set of segments to monitor. D_2 here refers to the second shortest path to D since D can be reached via multiple shortest paths. (a) $M_A : \{B, F\}$. (b) $M_C : \{F, D, E\}$. (c) $M_F : \{A, B, E, D_2\}$.

is subsumed by the m-routers. Furthermore, the set of measurement servers directly connected to nonupgraded routers is transformed into, what we call, m-servers. The m-servers are basically measurement servers that are capable of listening to the topology updates (OSPF LSAs) in the network, and are thus capable of reconstructing the forwarding paths in the network similar to the m-routers. For example, in Fig. 7, we show a toy topology with six routers connected via undirected edges and associated edge costs. Attached to each of the routers is a measurement server [shown in Fig. 7(a)] that issues data-plane probes to other such measurement servers to measure path properties of interest. In Fig. 7(b), we show the topology when two out of the six routers are upgraded to m-routers, and the measurement servers are converted to m-servers.

In the second step, each m-server or m-router identifies a set of nodes for which it monitors properties, called an *m-set*. It does so by first computing a self-sourced shortest-path spanning tree using Dijkstra's algorithm. The shortest-path trees computed at each of the six nodes are shown in Fig. 8. The m-router does not need to explicitly perform this computation and can leverage the existing shortest-path tree already computed by the OSPF process on the router. It then determines the m-set by making a cut in the tree whenever an m-router or an m-server is encountered. If an m-router is encountered, the rest of the paths to various destinations in the subtree of this m-router are monitored by that m-router. An m-server is encountered if no such m-router exists along the path (and hence it has to monitor this path itself). In Fig. 8, we show such m-sets for all the routers for the toy topology in Fig. 7. Note that in Fig. 8(c), the m-set consists of D_2 that corresponds to the second shortest path to D through E. The router F does not need to monitor the first shortest path to D through C.

Finally, m-router to m-router or m-server links (or virtual link consisting of paths through nonupgraded routers) are monitored using regular active probing. The m-routers report the internal measurements and the measurements to the nodes in the m-set

periodically to a monitoring station using specialized measurement state packets (MSPs). The m-servers only report the measurements to the nodes in the m-set within the MSP. The monitoring station uses the topology information to build n shortest-path trees (with each node as source) and uses the MSPs to directly diagnose any end-to-end problem in the network.

Our architecture also accommodates topology changes due to link failures or other reasons. Similar to what happens today, the m-routers (and the m-servers) recompute the new shortest paths when they receive OSPF LSAs and update their m-set by identifying the cut in the shortest-path tree again. During such periods, every m-router must continue to measure properties of the links or virtual links to the routers in both old and new m-sets for a configurable amount of time to ensure all paths are covered before and after reconvergence. Afterwards, they phase out the routers in the old m-set and restrict measurements to only those in the new one.

Advertising Presence of m-routers: Each m-router needs to identify the presence of other m-routers in the network in order to construct its m-set, for which we can leverage the existing OSPF protocol (or IS-IS) to allow m-routers to advertise their presence. We propose to use one of the reserved bits in the *Options* field of the OSPF control-plane messages for this task; the field exists precisely to advertise special capabilities of routers in the network.

A. Evaluating the Benefits

We now attempt to quantify the benefits achieved by incrementally deploying our architecture in real networks. Lacking access to actual tier-1 ISP topologies, we conduct our evaluation using the Rocketfuel topologies annotated with inferred link weights [34]. Despite the known deficiencies of this data, they suffice to demonstrate general trends. We compare the benefits of upgrading in a naive (random) fashion to an intelligent upgrade strategy.

We use a simple metric called *probe hop count* to quantitatively describe the benefit achieved by upgrading existing routers to m-routers. Probe hop count is defined as the sum of all the hops taken by every active probe that traverses the network. When active probes are issued from every measurement server to another, this translates to the sum of hop lengths of all the $O(n^2)$ shortest paths (including the multiple paths between a given pair of routers) in the network. On the other hand, in the mPLANE architecture, the probe hop count reduces to the total number of links in the network since each m-router transmits messages only to its adjacent routers. While the complexity of the probes is different in both the cases (active probes versus synchronization messages), we ignore this difference in this metric.

In order to identify candidate routers to upgrade, we guide the search in the direction of reducing the probe hop-count metric as much as possible. In particular, we select the routers that reduce the probe hop count the most. The exact algorithm is shown in Algorithm 1.

Fig. 9 shows the results of both upgrade strategies on the Rocketfuel autonomous system (AS) topology for Sprint network (results were similar on five other topologies we considered). From the graph, we can observe that the curve is convex

Algorithm 1: IdentifyRoutersToUpgrade(V, E, numUpdate)

```
1.  S = ComputeShortestPaths(V, E);
2.  U={ };
3.  numiter=0;
4.  while (numiter < numUpdate) do
5.     for path p ∈ S do
6.        for router r ∈ p − {src, dst} do
7.           count[r] ++;
8.        end for
9.     end for
10.    maxRouter = findMax(count)
11.    U = U +{maxRouter}
12.    for path p ∈ S do
13.       if maxRouter ∈ p then
14.          S = S − {p}
15.          p1= split p from src till maxRouter
16.          p2= split p from maxRouter till dst
17.          S = S + {p1, p2}
18.       end if
19.    end for
20.    numiter++;
21. end while
```

in shape; upgrading the first few routers results in maximum benefit, while the marginal benefit reduces drastically after a while. On average, upgrading about 15% of the routers in an intelligent fashion results in a two-orders-of-magnitude reduction in the probe hop count. For example, the Sprint topology in Fig. 9(a) requires approximately one million end-to-end active probes to measure each path without any upgraded routers. Upgrading 45 routers out of 315 results in a probe hop count of only 10 000—a two-orders-of-magnitude reduction in measurement overhead.

Fig. 9 shows the average localization granularity as well as maximum and minimum localization granularity, in terms of the size of an average segment. We can observe that the average localization granularity also drops down very rapidly (convex) for all the ISP topologies, indicating that upgrading a small number of routers can quickly achieve almost all the benefit. The benefit is achieved much slower, however, in the case of random upgrade. In particular, for the Sprint topology, upgrading about 1/6 of the routers (50 out of about 300) reduces the localization granularity to around 1.5 from 4. Of course, this is assuming only direct localization. If we were to couple this with other inference techniques, we can reduce this even further.

VII. Related Work

Traditionally, network operators determined link and hop properties using active measurement tools and inference algorithms. For example, the work by Chen *et al.* [6] and Duffield *et al.* [10] solve the problem of predicting the per-hop loss and latency characteristics based on end-to-end measurements (e.g., conducted using active probing tools [33], [23]) and routing information obtained from the network (e.g., using OSPF monitoring [32]). The advantages of our approach in comparison are twofold. First, LDA computes path and link properties by passively monitoring traffic in a router, so it

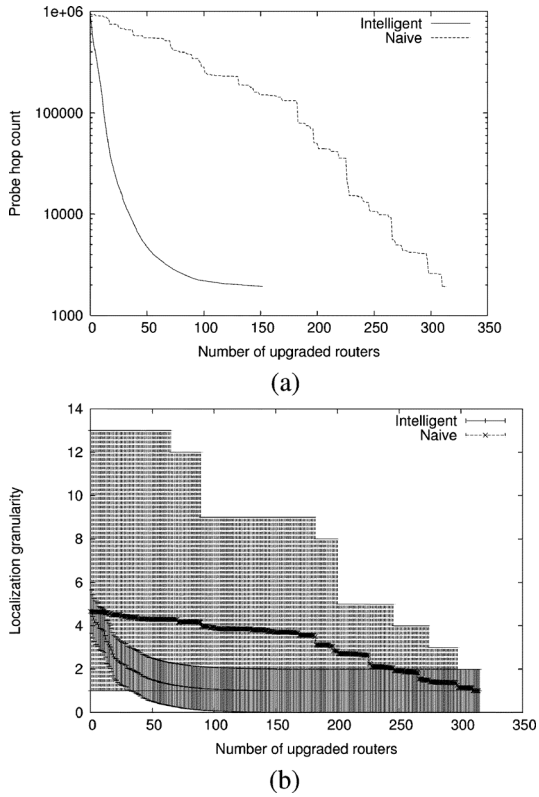


Fig. 9. (a) Probe bandwidth and (b) localization granularity for Sprint topology with increasing number of upgraded routers.

does not interfere with measurements or waste bandwidth by injecting any active probes. Second, LDA captures fine-grain latency measurements that can only be matched by extremely high frequency active probes (as discussed in Section IV-C). Furthermore, in our evaluation, we compared against localized active probes (i.e., between every pair of adjacent routers), which are more fine-grain than the current best practice (end-to-end probing) as it does not scale, requiring the monitoring of $O(m) \approx O(n^2)$ segments where m is the number of links, n is the number of routers.

We are not the first to suggest router extensions in support of fine-grain measurement. For example, Machiraju *et al.* argue for a measurement-friendly network architecture where individual routers provide separate priority levels for active probes [22]. Duffield *et al.* suggest the use of router support for sampling packet trajectories [8]. Passive measurement of loss and delay by directly comparing trajectory samples of the same packet observed at different points has been studied by Zseby *et al.* [40] and Duffield *et al.* [9]. Many high-speed router primitives have also been suggested in the literature for measuring flow statistics and detecting heavy-hitters [7], [11].

Papagiannaki *et al.* used GPS-synchronized (to microsecond accuracy) passive monitoring cards to trace all packets entering and leaving a Sprint backbone router [27]. Each packet generates a fixed-size timestamped record, allowing exact delays, as well as other statistics, to be computed to within clock accuracy. From a measurement standpoint, their approach represents the ideal: exact packet-for-packet accounting. Unfortunately, as they themselves point out, such an approach is "computationally intensive and demanding in terms of storage," making widespread production deployment infeasible. Hohn *et al.* describe a mechanism to obtain router delay information using the amplitude and duration of busy periods [14]. While their approach provides only an approximate distribution, it can be effective in determining the order of magnitude of delay.

VIII. Conclusion

This paper proposes a mechanism that vendors can embed directly in routers to cheaply provide fine-grain delay and loss measurement. Starting from the simple idea of keeping a sum of sent timestamps and a sum of receive timestamps that is not resilient to loss, we developed a strategy to cope with loss using multiple hash buckets, and multiple sampling granularities to deal with unknown loss values. Furthermore, we adapt the classic approach to L2-norm estimation in a single stream to also calculate the standard deviation of delay. Loss estimation, of course, falls out trivially from these data structures. We emphasize that our mechanism complements—but does not replace—end-to-end probes. Customers will continue to use end-to-end probes to monitor the end-to-end performance of their applications. Furthermore, it is unlikely that LDA will be deployed at all links along many paths in the near future. However, LDA probes can proactively discover latency issues, especially at very fine scales, thata network manager can then address. Moreover, if an end-to-end probe detects a problem, a manager can use the LDA mechanism on routers along the path to better localize the problem.

References

[1] Cisco, "Cisco extends highly secure, improved communications in rugged environments with new family of industrial Ethernet switches," 2007 [Online]. Available: http://newsroom.cisco.com/dlls/2007/prod_111407.html
[2] Corvil, Ltd., "Corvil, Ltd.," 2011 [Online]. Available: http://www.corvil.com
[3] N. Duffield, V. Paxson, and D. Towsley, "MINC—Multicast-based inference of network-internal characteristics," [Online]. Available: http://gaia.cs.umass.edu/minc/
[4] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 137–147, Feb. 1999.
[5] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003," in *Proc. ACM SIGCOMM Workshop Netw. Games*, Aug. 2004, pp. 144–151.
[6] Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An algebraic approach to practical and scalable overlay network monitoring," in *Proc. ACM SIGCOMM*, Sep. 2004, pp. 55–66.
[7] A. Dobra, M. Garofalakis, J. E. Gehrke, and R. Rastogi, "Processing complex aggregate queries over data streams," in *Proc. ACM SIGMOD*, Jun. 2002, pp. 61–72.
[8] N. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," in *Proc. ACM SIGCOMM*, Aug. 2000, pp. 271–282.
[9] N. Duffield, A. Gerber, and M. Grossglauser, "Trajectory engine: A backend for trajectory sampling," in *Pro. IEEE Netw. Oper. Manage. Symp.*, Apr. 2002, pp. 437–450.
[10] N. Duffield, "Simple network performance tomography," in *Proc. USENIX/ACM Internet Meas. Conf.*, Oct. 2003, pp. 210–215.
[11] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, Aug. 2003.
[12] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Pro. USENIX/ACM Internet Meas. Conf.*, Oct. 2003, pp. 153–166.

[13] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 13–30, Mar. 1963.
[14] N. Hohn, D. Veitch, K. Papagiannaki, and C. Diot, "Bridging router performance and queuing theory," in *Proc. ACM SIGMETRICS*, Jun. 2004, pp. 355–366.
[15] *Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE/ANSI 1588 Standard, 2002.
[16] Fibre Channel Backbone-5 (FC-BB-5). ver. 1.03, INCITS, Oct. 2008.
[17] S. Kandula, D. Katabi, and J. P. Vasseur, "Shrink: A tool for failure diagnosis in IP networks," in *Proc. ACM SIGCOMM MineNet*, Aug. 2005, pp. 173–178.
[18] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "Detection and localization of network black holes," in *Proc. IEEE INFOCOM*, Anchorage, AK, May 2007, pp. 2180–2188.
[19] A. Kumar, M. Sung, J. Xu, and E. W. Zegura, "A data streaming algorithm for estimating subpopulation flow size distribution," in *Proc. ACM SIGMETRICS*, Jun. 2005, pp. 61–72.
[20] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," in *Proc. ACM SIGMETRICS*, Jun. 2006, pp. 145–156.
[21] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: A novel counter architecture for per-flow measurement," in *Proc. ACM SIGMETRICS*, Jun. 2008, pp. 121–132.
[22] S. Machiraju and D. Veitch, "A measurement-friendly network (MFN) architecture," in *Proc. ACM SIGCOMM Workshop Internet Netw. Manage.*, Sep. 2006, pp. 53–58.
[23] J. Mahdavi, V. Paxson, A. Adams, and M. Mathis, "Creating a scalable architecture for Internet measurement," in *Proc. INET*, Jul. 1998.
[24] R. Martin, "Wall street's quest to process data at the speed of light," *InformationWeek* 2007 [Online]. Available: http://www.informationweek.com/news/infrastructure/showArticle.jhtml?articleID=199200297
[25] V. Misra, W.-B. Gong, and D. Towsley, "Stochastic differential equation modeling and analysis of TCP windowsize behavior," in *Proc. IFIP WG 7.3 Perform.*, Nov. 1999.
[26] H. X. Nguyen and P. Thiran, "Network loss inference with second order statistics of end-to-end flows," in *Proc. ACM SIGCOMM Internet Meas. Conf.*, Oct. 2007, pp. 227–240.
[27] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot, "Measurement and analysis of single-hop delay on an IP backbone network," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 6, pp. 908–921, Aug. 2003.
[28] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Trans. Comput.*, vol. 46, no. 12, pp. 1378–1381, Dec. 1997.
[29] M. Riska, D. Malik, and A. Kessler, "Trading flow architecture," Cisco Systems, Inc., San Jose, CA, Tech. Rep., 2008 [Online]. Available: http://www.cisco.com/en/US/docs/solutions/Verticals/Trading_Floor_Architecture-E.pdf
[30] S. Savage, "Sting: A TCP-based network measurement tool," in *Proc. USENIX Symp. Internet Technol. Syst.*, Oct. 1999, vol. 2, p. 2.
[31] M. Saxena and R. R. Kompella, "On the inadequacy of link-connectivity monitoring," in *Proc. IEEE Workshop Autom. Netw. Manage.*, Phoenix, AZ, Apr. 2008, pp. 1–6.
[32] A. Shaikh and A. Greenberg, "OSPF monitoring: Architecture, design and deployment experience," in *Proc. USENIX NSDI*, Mar. 2004, p. 5.
[33] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Improving accuracy in end-to-end packet loss measurement," in *Proc. ACM SIGCOMM*, Aug. 2005, pp. 157–168.
[34] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *Proc. ACM SIGCOMM*, 2002, pp. 133–145.
[35] T. Szigeti and C. Hattingh, "Quality of service design overview," Cisco, San Jose, CA, Dec. 2004 [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=357102&seqNum=2
[36] F. Toomey, "Monitoring and analysis of traffic for low-latency trading networks," Corvil, Ltd., New York, NY, Tech. Rep., 2008.
[37] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *J. Amer. Statist. Assoc.*, vol. 91, pp. 365–377, 1996.
[38] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast accurate computation of large-scale IP traffic matrices from link loads," in *Proc. ACM SIGMETRICS*, Jun. 2003, pp. 206–217.
[39] Y. Zhao, Y. Chen, and D. Bindel, "Towards unbiased end-to-end network diagnosis," in *Proc. ACM SIGCOMM*, Sep. 2006, pp. 219–230.
[40] T. Zseby, S. Zander, and G. Carle, "Evaluation of building blocks for passive one-way-delay measurements," in *Proc. Passive Active Meas. Workshop*, Apr. 2001 [Online]. Available: http://caia.swin.edu.au/cv/szander/publications/zseby01evaluation.pdf

Ramana Rao Kompella (M'07) received the B.Tech. degree from the Indian Institute of Technology (IIT), Bombay, India, in 1999., the M.S. degree from Stanford University, Stanford, CA, in 2001, and the Ph.D. degree from the University of California, San Diego (UCSD), in 2007, all in computer science and engineering.

He is currently an Assistant Professor with the Department of Computer Sciences, Purdue University, West Lafayette, IN. His main research interests include routing and transport issues in datacenter networks and virtualized clouds, fault localization in networked systems, scalable measurement techniques for high-speed switches and routers, and scheduling in wireless networks.

Dr. Kompella has been a member of the Association for Computing Machinery (ACM) since 2007. He received the NSF CAREER Award in 2011.

Kirill Levchenko studied computer science and mathematics at the University of Illinois at Urbana–Champaign. He received the Ph.D. degree in computer science from the University of California, San Diego, in 2009.

He is currently a Postdoctoral Scholar with the University of California, San Diego. He research interests include network routing and network security, with a focus on e-crime and underground economies.

Alex C. Snoeren (S'00–M'03) received the B.S. degrees in computer science and applied mathematics and M.S. degree in computer science from the Georgia Institute of Technology, Atlanta, in 1996, 1997, and 1997, respectively, and the Ph.D. degree in computer science from the Massachusetts Institute of Technology (MIT) in 2003.

He is an Associate Professor with the Computer Science and Engineering Department, University of California, San Diego. His research interests include operating systems, distributed computing, and mobile and wide-area networking.

George Varghese (M'99) received the Ph.D. degree in computer science from the Massachusetts Institute of Technology, Cambridge, in 1992.

He is currently a Professor of computer science with the University of California, San Diego (UCSD). In May 2004, he cofounded NetSift, Inc., San Diego, CA, where he was the President and CTO. NetSift is now part of Cisco Systems. From August 2005 to August 2006, he worked with Cisco Systems to help equip future routers and switches to detect traffic patterns to facilitate traffic measurement and real-time intrusion detection.

Prof. Varghese won the ONR Young Investigator Award in 1996 and was elected to be a Fellow of the Association for Computing Machinery (ACM) in 2002.