Every Microsecond Counts: Tracking Fine-Grain Latencies with a Lossy Difference Aggregator

Ramana Rao Kompella[†], Kirill Levchenko, Alex C. Snoeren, and George Varghese

[†]Purdue University and University of California, San Diego

ABSTRACT

Many network applications have stringent end-to-end latency requirements, including VoIP and interactive video conferencing, automated trading, and high-performance computing—where even microsecond variations may be intolerable. The resulting fine-grain 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 and losses as infrequent as one in a million.

Such measurement can be viewed abstractly as what we refer to as a *coordinated streaming* problem, which is fundamentally harder than standard streaming problems due to the need to coordinate values between nodes. We describe a compact data structure that efficiently computes the average and standard deviation of latency and loss rate in a coordinated streaming environment. Our theoretical results translate to an efficient hardware implementation at 40 Gbps using less than 1% of a typical 65-nm 400-MHz networking ASIC. 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.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network management

General Terms

Measurement, algorithms

Keywords

Passive measurement, packet sampling

1. INTRODUCTION

An increasing number of Internet-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. These applications range from popular multimedia services like voice-over-IP, multi-player gaming, and video conferencing to niche—but commercially important—markets like automated trading and high-performance computing. As such applications grow in significance, customers are placing increasing demands on operators to provision and manage 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 variations or loss spikes. Hence, we propose a new mechanism to measure latency and loss at extremely small time scales, even tens of microseconds. We focus particularly on data-center networks, where such measurement is both useful (because many data-center applications can be hurt if latencies increase by even tens of microseconds) and feasible (because of small propagation delays). Our approach can also be applied more generally, but the accuracy of existing techniques may be sufficient for many wide-area scenarios.

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 microseconds of each other [35]. (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" [25].)

Current routers typically support two distinct accounting mechanisms: SNMP and NetFlow. Neither are up to the task. SNMP provides only cumulative counters which, 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 [10]). 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 [24, 31, 33]. Latency estimates computed in this fashion, however, can be grossly inaccurate in practice. In a recent Cisco study, periodic

SIGCOMM'09, August 17–21, 2009, Barcelona, Spain.

probes sent at 1-second 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 [30, Fig. 6]. Capturing these effects in real networks requires injecting a prohibitively high rate of probe packets. For these reasons, operators often employ external passive hardware monitors (e.g., those manufactured by Corvil [1]) 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 time scales 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 4.3.

- *Low overhead:* Our suggested 40-Gbps LDA implementation uses less than 1% of a standard networking ASIC and 72 Kbits of control traffic per second, as detailed in Section 5.1.

- *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 5.1.

- *Assists fault localization:* LDA can operate link-by-link and even segment-by-segment within a router, enabling direct, precise performance fault localization. Section 5.2 describes a potential fault-localization system based upon LDA.

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, next-generation 65-nm router chips have a large number of (currently unallocated) transistors. Third, 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.

2. PRELIMINARIES

A number of commercially important network applications have stringent latency demands. Here, we describe three such domains and provide concrete requirements to drive our design and evaluation of LDA. In addition, we present an abstract model of the measurement task, an instance of what we term a coordinated streaming problem. We then show that some coordinated problems have fundamentally high space complexity as compared to traditional streaming versions of the same problem.

2.1 Requirements

An application's latency requirements depend greatly on its intended deployment scenario. We start by considering the specific requirements of each domain in turn, and then identify the overall measurement metrics of interest.

2.1.1 Domains

Wide-area multi-media applications have demands on the order of a few hundred milliseconds (100–250 ms). Latency requirements in the financial sector are tighter (100 μs—1 ms), but the most stringent requirements are in cluster-based high-performance computing which can require latencies as low as 1–10 μs.

Interactive multi-media: Games that require fast-paced interaction such as World of WarCraft or even first-person shooter games like Quake 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]. There has also been a large increase in voice-over-IP (VoIP) and interactive video. While pre-recorded material can be buffered, interactive applications like 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 [34].

Automated trading: Orders in financial markets are predominantly placed by machines running automatic trading algorithms that respond to quotes arriving from exchanges. For example, in September 2008, the London Stock Exchange announced a service that provided frequent algorithmic trading firms with sub-millisecond access to market data [21]. Because machines—not people—are responding to changing financial data (a recent survey indicates that 60–70% of the trades on the NYSE are conducted electronically, and half of those are algorithmic [25]), delays larger than 100 microseconds can lead to arbitrage opportunities that can be leveraged to produce large financial gains.

Trading networks frequently connect market data distributors to trading floors via Ethernet-switched networks. A typical problem encountered while maintaining these networks is a saturated core 1-Gbps link that increases latency by 10s of microseconds. The situation could be addressed (at significant cost) by upgrading the link to 10-Gbps, but only after the overloaded link is detected and isolated. Hence, a market has emerged for sub-microsecond measurement. For example, Cisco resells a passive device manufactured by Corvil [30] that can detect microsecond differences in latency [35].

High-performance computing: The move from supercomputing to cluster computing has placed increased demands on data-center networks. Today, Infiniband is the de-facto interconnect for high-performance clusters and offers latencies of one microsecond or less across an individual switch and ten microseconds 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. Back-end storage-area networks have similar demands, where Fiber Channel has emerged to deliver similar latencies between CPUs and remote disks, replacing the traditional I/O bus.

As a result, many machines in high-performance data centers are connected to Ethernet, Infiniband, and Fiber Channel networks. Industry is moving to integrate these disparate technologies using commodity Ethernet switches through standards such as Fibre Channel over Ethernet [16]. Hence, the underlying Ethernet networks must evolve to meet the same stringent delay requirements, on the order of tens of microseconds, as shown in the specifications of recent Ethernet switch offerings from companies like Woven Systems [37] and Arista [4].

2.1.2 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

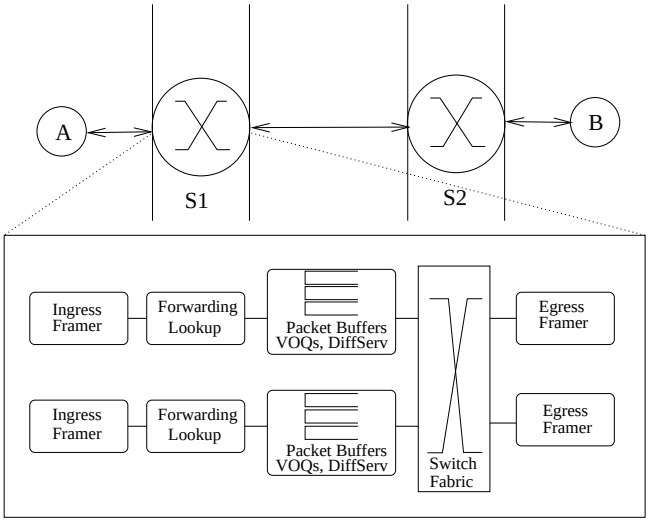


Figure 1: An path decomposed into measurement segments.

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.)

2.2 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 [24, 31, 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 [2, 6, 8, 17, 27, 36, 38] to isolate the latency of individual segments [18, 39], we propose to instrument each segment of the network with our new measurement primitive. (We will return to consider incremental deployment issues in Section 5.2.) Thus, in our model, every end-to-end path can be broken up into what we call *measurement segments*. For example, as shown in Figure 1, a 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 10s of microseconds. Such a router segment can be further decomposed as shown in the bottom of Figure 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.

Thus, we focus on a single measurement segment between a sender A and a receiver B. The segment could be a single link, or may consist of a (set of) internal path(s) within a router that contains packet queues. We assume that the segment provides FIFO packet delivery. In practice, packets are commonly load balanced across multiple component links resulting in non-FIFO behavior overall, but in that case we assume that measurement is conducted at resequencing points or separately on each component link. We further assume that the segment endpoints are tightly time synchronized (to within a few microseconds). If the clocks at sender and receiver differ by e, then all latency estimates will have an additive error of e as well.

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 which 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].

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 synposis S to the receiver in the End Control Message. This allows the receiver to compute latency and loss estimates as some function of S and R.

Note that the receiver can start much later than the sender if the Start Control message takes a long time, but the goal is merely that the sender and receiver compute the synopses over the same set of packets. This is 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 control packets. 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 a periodic 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; 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. Further, 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.

2.3 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 naïve 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 [28]. Unfortunately, the naïve 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 one second, and the segment operates at 40 Gbps, then N can be as large as 125 million 40-byte 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, 22]. 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, Matias and Szegedy [3].) Despite this negative result for the maximum delay, we will show that approximating both average and standard deviation of delay can be done efficiently. In the next section, we describe the Lossy Difference Aggregator, a mechanism that estimates these statistics.

3. LDA

A Lossy Difference Aggregator (LDA) 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 second—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 (which is required by all one-way delay measurement mechanisms) and consistent packet ordering at the sender and receiver.

3.1 The data structure

To better explain the LDA, we begin with the simplest average delay measurement primitive—a pair of counters—and then develop the full LDA as shown in Figure 3.

3.1.1 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.

3.1.2 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 one of the m accumulator-counter pairs, and the corresponding timestamp accumulator and packet counter are updated as before. By using the same 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 sender's packet counter at the index of the lost packet will be one more than the corresponding packet counter on the receiver. We call such an index *unusable* and do not use it in calculating our average delay estimate. The remaining usable 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 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. Figure 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 accumulator pairs is $(180 - 120) + (37 - 15) + (14 - 6) = 90$; the effective sample size is $5 + 2 + 1 = 8$. The estimated delay is thus $90/8 = 11.25$.

3.1.3 Known loss rate

For larger loss rates, we need to sample the incoming packets to reduce the number of potentially unusable rows. Sampling can

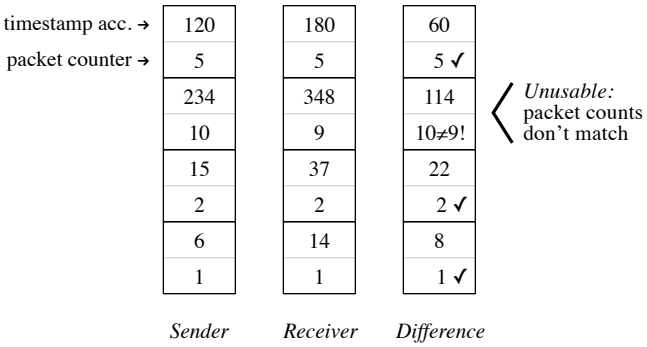


Figure 2: Computing LDA average delay with one bank of four timestamp accumulator-counter pairs. Three pairs are usable (with 5, 2, and 1 packets), while the second is not due to a packet loss. Thus, the average delay is $(60 + 22 + 8)/(5 + 2 + 1)$.

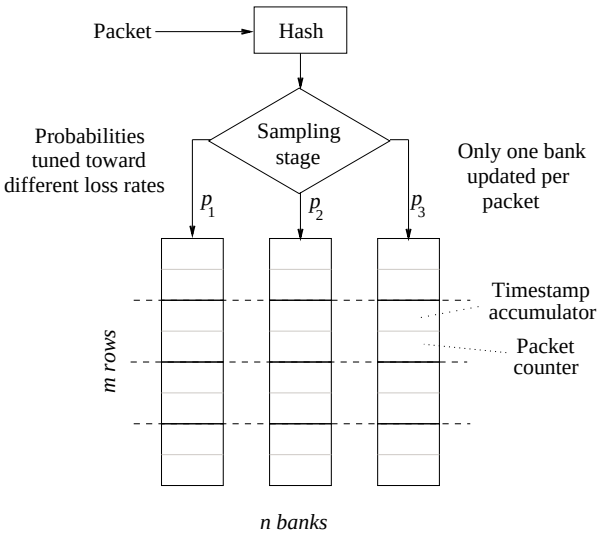


Figure 3: The Lossy Difference Aggregator (LDA) with n banks of m rows each.

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 3.3 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.

3.1.4 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 (Figure 3). (In our experiments, we found that two banks are a reasonable choice.)

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 multi-bank 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 *non-uniformly* according to bank sampling probabilities $p_1, p_2, \ldots, p_n$. For non-uniform 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 useable elements across all banks. The full $m \times n$ LDA is shown in Figure 3.

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.

3.2 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 $g(x) = 0$, the 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:

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**

3.3 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 don't, 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

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

T_A and T_B are the sum of the of the useable 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 and $L \geq m$, 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 $\alpha = 0.5$, the value we use in our experiments.

3.4 Latency standard deviation

Note that we exploited the fact that the sum of the differences of receive and send packet time stamps 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's 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

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

but we also get unwanted cross terms. Fortunately, the cross terms can be eliminated using a technique introduced by Alon, Matias and Szegedy [3]. 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:

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

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

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

It is straightforward to show that

$$\mathrm{E}[F] = \frac{1}{R}\sum_x (b_x - a_x)^2$$

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

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

For comparison, the basic estimator (4), which does not handle packet loss, has variance

$$\frac{2}{R^2} \sum_{x \neq x'}^{\text{rec'd}} w_x^2 w_{x'}^2.$$

By averaging several instances of the estimator as in [3], the variance can be reduced arbitrarily. In our experiments, however, we use the estimator (5) directly with satisfactory results. It is worth remembering that this standard deviation estimate comes "for free" by mining the LDA data structure (designed for estimating average) for more information.

4. 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 [35], 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 with 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 [28], 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 to 5 so that the delay values do not become too skewed and to ensure that the distributions have bounded variance.

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's 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 with 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 [26]), where each episode involves dropping a burst of packets (following the model of Sommers *et al.* [33]).

4.1 Validation

The main goal of the set of experiments described in this subsection 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 $m = 1,024$ counters. We simulate a 10-Gbps OC-192 link which, assuming an average packet size of 250 bytes, carries roughly five million packets per second at capacity. (The choice of 250-byte packets is arbitrary and results in round numbers; the functionality of LDA is not impacted by packet size.) We simulate a measurement interval of one second (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.

[1]The main advantage of standard packages like ns2 is the library of prexisting 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 speed up of a custom solution.

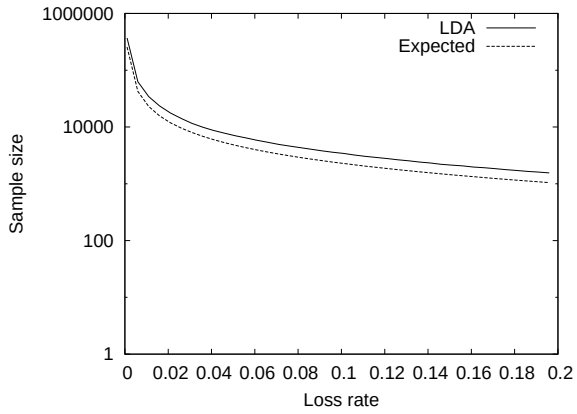


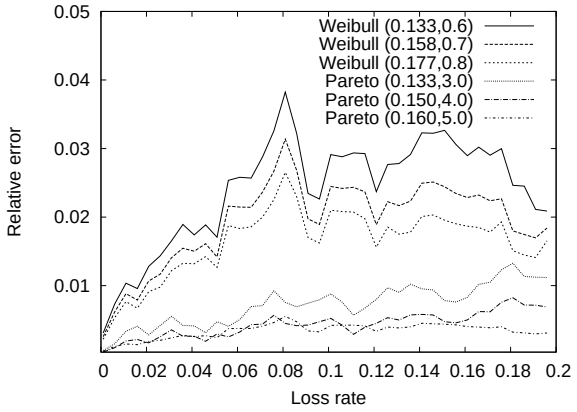
Figure 4: The sample size obtained by a single-bank LDA as a function of loss rate.

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.

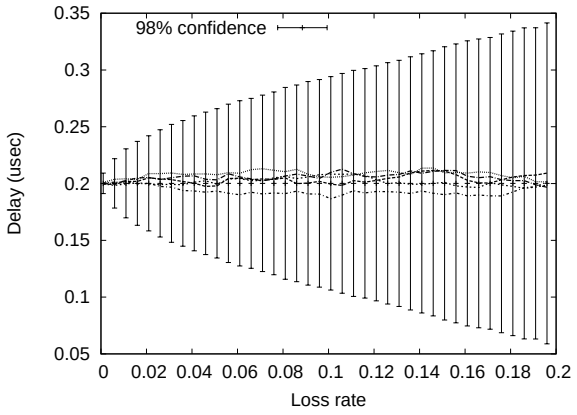
We first verify empirically that the actual sample size obtained using our data structure matches expectation. For the purposes of this experiment, we assume that we know *a priori* the loss rate l; we compute the number of lost packets $L = N \cdot l$ and set the sampling probability accordingly as $p = \alpha m/(L+1)$, where $\alpha = 0.5$.

Figure 4 shows the number of samples captured by the LDA as we vary the loss rate from 0.5% to 20%, as well as the expected value given by Equation 2. Two main observations can be made from the figure: First, as expected, the sample size decreases as loss rate increases. Second, our analytical bound is conservative; LDA captures more samples in simulation than according to theory.

In Figure 5(a), we plot the average relative error (defined as $|true - estimated|/|true|$) 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 ten 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 subsection. 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%.



(a) Average relative error



(b) Estimated average delay

Figure 5: Average relative error and 98% confidence bounds of the delay estimates 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.

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 Figure 5(b), rather than report relative error, we graph the actual delay estimate computed by a representative five of the ten constituent runs in Figure 5(a). In addition, we plot the 98%-confidence bounds computed using Equation 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 ten runs from Figure 5(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 Figure 6. 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

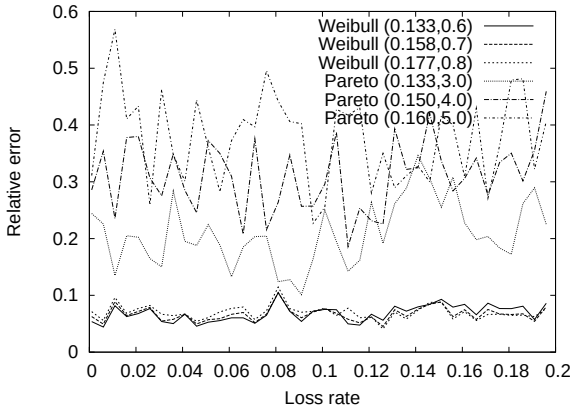


Figure 6: Average relative error of LDA's standard-deviation estimator as a function of loss rate.

Pareto- and Weibull-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 [28], so relative error is likely to be small in practice.

4.2 Handling unknown loss rates

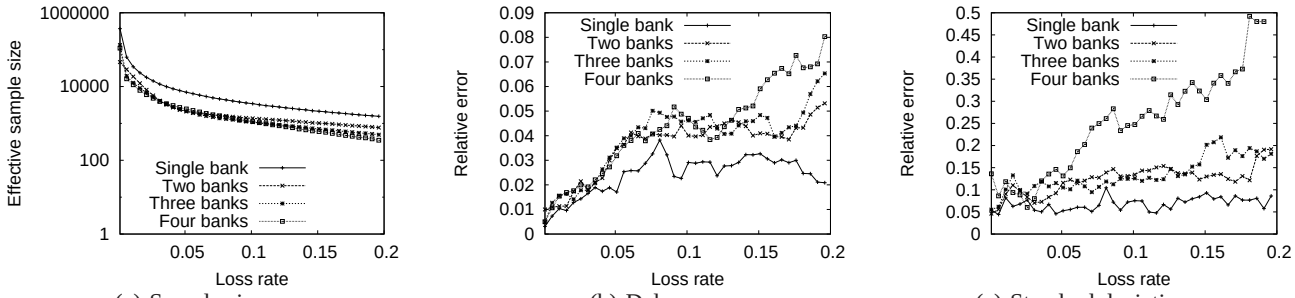
Up to this point, we have configured each LDA optimally for the actual loss rate. Obviously, any real deployment will need to be configured for a range of loss rates. Here, we evaluate the efficacy of various configurations of multi-bank LDAs over a range of loss rates. In particular, each bank within 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 (1,024) of counters: two banks of 512 counters tuned towards loss rates of 0.005 and 0.1, three banks with roughly one-third of the counters tuned towards 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 Figure 7. 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 Figures 5 and 6.

Figure 7(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 Figures 7(b) and 7(c) 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 towards 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.

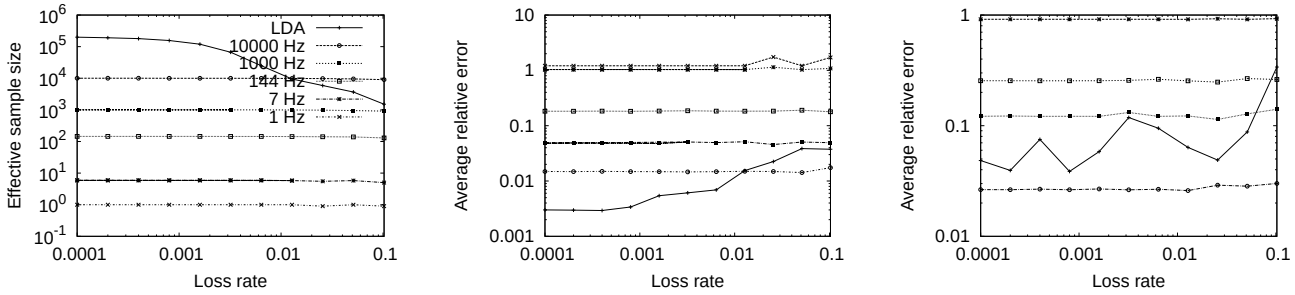
4.3 Comparison with active probes

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 [24]) for various probe frequencies. The ac-



(a) Sample size (b) Delay (c) Standard deviation

Figure 7: The performance of various multi-bank LDA configurations.



(a) Sample size (b) Delay (c) Standard deviation

Figure 8: Sample size, delay and standard deviation estimates obtained using a two-bank LDA in comparison with active probing at various frequencies. Log-scale axes.

curacy 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 1,024 $\times$ 72 bits $\approx$ 72 Kbits for the data structure and 1,500-byte 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 bytes). 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.

We generate Poisson-modulated active probes by injecting probe packets at intervals distributed according to a Poisson process with the desired average inter-arrival 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. We plot two sets of graphs. First, in Figure 8, 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. In Figure 9, we show the confidence bounds obtained for LDA. (We refrain from presenting confidence intervals for active probes, as the bounds that can be derived are unfairly loose.)

Figure 8(a) clearly shows the impact of increased probe frequency: more samples. As before, each point represents the average of ten 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 Figure 8(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% vs. $\approx$ 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. Figure 9 shows that the 98%-confidence intervals for the constituent LDA runs from Figure 8(a) are quite small—generally

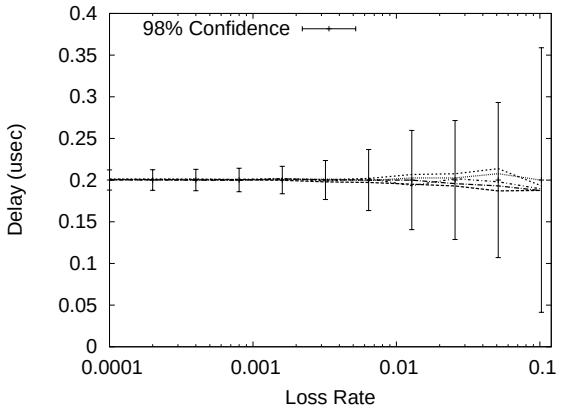


Figure 9: Delay estimates and 98% confidence bounds from a two-bank LDA. Actual mean delay is 0.2 μs.

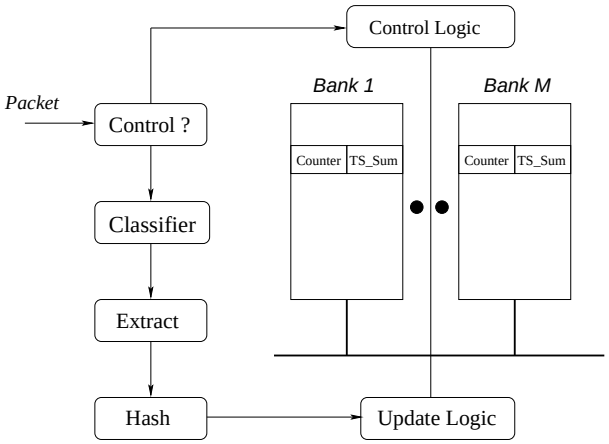


Figure 10: Potential LDA chip schematic

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%.

5. REALIZATION

In this section, we discus two separate aspects of LDA deployment. First, we explain how LDAs can be implemented within a router by presenting a sample design. Then, we describe how LDAs could be deployed across routers (i.e., to measure links) and used for fault localization.

5.1 Hardware implementation

We have sketched out the base logic for LDA that we estimate takes less than 1% of a low-end 10 mm $\times$ 10 mm networking ASIC, based on the standard 400-MHz 65-nm process currently being employed by most networking chipset vendors. The logic is flow-through—in other words, it can be inserted into the path of a link between the sender and receiver end without changing any other logic—allowing LDA to be incrementally deployed within existing router designs.

A minimal implementation would place a single LDA together with MAC logic at ingress and egress links; ingress-egress paths pass through a number of points within a router where packets can be queued and, therefore, delayed or lost. We envision, however, that deployed LDAs will contain a packet classifier that identifies a particular class of traffic, such as flows to a specified TCP port. Hence, it may be useful to include multiple LDAs on a single line card. Most mid-range ASICs should be able to afford up to ten separate copies of the LDA logic to handle various traffic classes and/or priority levels. Finally, we observe that while many routers internally stripe packets at various points in the router which would break our FIFO assumption, packets are resequenced at various points. Thus, rather than placing LDA logic on every striped path, it may be cheaper to place the LDA receiver logic where resequencing takes place.

Figure 10 shows a schematic of our initial design. The logic at sender and receiver is nearly identical. At the sender (receiver) the first X bytes of the packet—say fifty—are sent to the logic. The logic first determines if it is a control or data packet using, say, an Ethernet type field.

If the received packet is a data packet, a classifier is used to select the type of packet being measured. The update logic extracts some fixed bytes from the packet (say bytes 50–100) and computes a hash. H3 hash functions [29], 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 hash output supplies a 64-bit number which is passed to the update logic. The high-order bits select the sampling probability which in turn determines which bank is selected. For example, if there are two banks, that are selected with probabilities 1/2 and 1/64, the six high-order bits are used. If the first six bits are zero, the second bank is selected; if the first six bits are non-zero and the first bit is zero, the first bank is selected.

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 1,024 counters, we use the ten 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 time stamp 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 translate 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 which computes delay, loss, and variance estimates in software which it can then report to a management station on demand.

The control logic can work in two ways. The simplest way is to keep two copies of each counter so that the control logic 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 can be used: one to record the end of an interval, and a second control packet sent T' seconds later to denote the start of the next interval. During the intervening period, the update logic is disabled to allow the control logic to read all counters. The disadvantage is that though T' can be small, 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, 1,000

SRAM counters of 72 bits each takes 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 mm $\times$ 10 mm) ASICs on the market today.

5.2 Deployment and fault localization

The easiest path to deployment is to first deploy within individual routers where the majority of loss and delay occur. It may also be useful to deploy across links because of optical device (e.g., SONET) reconfigurations and degradations. The difficulty with deploying across links is the need for microsecond precision and the need for a protocol change. Fortunately, a solution to both problems can be found in terms of a new precision time-synchronization standard called IEEE 1588 [15] being deployed by major router vendors. IEEE 1588 uses synchronization messages that are intercepted by hardware. IEEE 1588 can easily be extended to handle LDA using a few extra control message types and the logic described above.

Significant benefits can be derived from a full deployment, where LDAs are deployed at each and every router and link. In particular, performance fault localization—traditionally a very challenging problem [18, 39]—becomes straightforward. We envision the presence of a centralized monitoring station which could use a topology monitor (such as OSPF monitor [32]) to decompose a misbehaving end-to-end path into segments, and query each segment to isolate the misbehaving (e.g., high-delay) segment. Scaling to hundreds or even thousands of collectors seems straightforward, as each summary structure is only a few kilobits in size. Even maintaining one-second intervals—which may be overkill for large deployments—the bandwidth requirement at the collection point would be on the order of a megabit per second for a thousand measurement points.

Even stopping short of pervasive deployment, LDA can be extended to include virtual links between pairs of upgraded routers, in an overlay topology consisting of just upgraded routers connected via legacy router hops. We omit the details for lack of space, but our experiments with RocketFuel topologies show that upgrading 1/6th of the routers in the Sprint topology reduces the localization granularity (the average path length between upgraded routers) to around 1.5.

6. 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.* [8] 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, 24]) and routing information obtained from the network (e.g., using OSPF monitoring [32]). The advantages of our approach in comparison are two fold. First, LDA computes path and link properties by passively monitoring traffic in a router, so it does not interfere with measurements or waste bandwidth by injecting any active probes. Second, LDA captures fine-grain latency measurements that can be only be matched by extremely high frequency active probes (as discussed in Section 4.3). Further, 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 [23]. Duffield *et al.* suggest the use of router support for sampling packet trajectories [10]. 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 [28]. Each packet generates a fixed-size time-stamped 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 wide-spread 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.

7. 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 which 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. Further, 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. Further, 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, that a network manager can then address. Further, 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.

While our setting begs comparisons to streaming, we introduce a new streaming problem: two-party coordinated streaming with loss. In this setting, problems that were trivial in the single-party streaming setting (such as identifying the maximum value) are now provably hard. Thus, we believe coordinated streaming may be an interesting research area in its own right: Which coordinated functions can be computed with low memory? Further, there are functions which would be useful in practice (e.g., loss distributions) that we do not yet know how to compute efficiently.

From a router-vendor standpoint, the efficiency of the proposed technique seems acceptable. Moreover, we observe that all microchips today have a component called JTAG whose overhead chip vendors happily pay for the benefit of increased ease of configuration and debugging. Our broader vision is that all networking

chips should also have a small "MTAG" component to facilitate fine-grain measurement of latency and loss. The LDA primitives described in this paper would be a candidate for such an MTAG component. With such a component universally deployed, the network manager of the future could pin-point loss spikes anywhere in the networking path of a critical network application with microsecond accuracy.

Acknowledgments

Hilary Finucane first observed that arbitrary columns of the LDA can be summed to compute a more accurate estimate. In addition, the authors are indebted to John Huber of Cisco Systems for providing sizing information critical to our hardware design; and Michael Mitzenmacher, Subhash Suri, the anonymous reviewers, and Darryl Veitch, our shepherd, for comments on previous versions of this manuscript. This work was supported in part by NSF awards CNS-0347949, CNS-0831647, and a grant from Cisco Systems.

8. REFERENCES

[1] Corvil, Ltd. `http://www.corvil.com`.

[2] Multicast-based intference of network-internal characteristics. `http://gaia.cs.umass.edu/minc/`.

[3] ALON, N., MATIAS, Y., AND SZEGEDY, M. The space complexity of approximating the frequency moments. *J. Computer and System Sciences 58*, 1 (Feb. 1999), 137–147.

[4] ARISTA NETWORKS, INC. 7100 series datasheet. `http://www.aristanetworks.com/en/7100_datasheet.pdf`, 2008.

[5] BEIGBEDER, T., COUGHLAN, R., LUSHER, C., PLUNKETT, J., AGU, E., AND CLAYPOOL, M. The effects of loss and latency on user performance in Unreal Tournament 2003. In *Proceedings of the ACM SIGCOMM Workshop on Network Games* (Aug. 2004).

[6] CHEN, Y., BINDEL, D., SONG, H., AND KATZ, R. H. An algebraic approach to practical and scalable overlay network monitoring. In *ACM SIGCOMM* (Sept. 2004).

[7] DOBRA, A., GAROFALAKIS, M., GEHRKE, J. E., AND RASTOGI, R. Processing complex aggregate queries over data streams. In *Proceedings of ACM SIGMOD* (June 2002).

[8] DUFFIELD, N. Simple network performance tomography. In *Proceedings of USENIX/ACM Internet Measurement Conference* (Oct. 2003).

[9] DUFFIELD, N., GERBER, A., AND GROSSGLAUSER, M. Trajectory engine: A backend for trajectory sampling. In *Proceedings of IEEE Network Operations and Management Symposium* (Apr. 2002).

[10] DUFFIELD, N., AND GROSSGLAUSER, M. Trajectory sampling for direct traffic observation. In *Proceedings of ACM SIGCOMM* (Aug. 2000).

[11] ESTAN, C., AND VARGHESE, G. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems 21*, 3 (Aug. 2003).

[12] ESTAN, C., VARGHESE, G., AND FISK, M. Bitmap algorithms for counting active flows on high speed links. In *Proceedings of the USENIX/ACM Internet Measurement Conference* (Oct. 2003).

[13] HOEFFDING, W. Probability inequalities for sums of bounded random variables. *J. American Statistical Association 58*, 301 (March 1963), 13–30.

[14] HOHN, N., VEITCH, D., PAPAGIANNAKI, K., AND DIOT, C. Bridging router performance and queuing theory. In *Proceedings of ACM SIGMETRICS* (June 2004).

[15] IEEE. Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2002. IEEE/ANSI 1588 Standard.

[16] INCITS. Fibre channel backbone-5 (FC-BB-5), Oct. 2008. Ver. 1.03.

[17] KANDULA, S., KATABI, D., AND VASSEUR, J. P. Shrink: A tool for failure diagnosis in IP networks. In *Proceedings of ACM SIGCOMM MineNet Workshop* (Aug. 2005).

[18] KOMPELLA, R. R., YATES, J., GREENBERG, A., AND SNOEREN, A. C. Detection and localization of network black holes. In *Proceedings of IEEE Infocom* (May 2007).

[19] KUMAR, A., SUNG, M., XU, J., AND ZEGURA, E. W. A data streaming algorithm for estimating subpopulation flow size distribution. In *Proceedings of ACM SIGMETRICS* (June 2005).

[20] LALL, A., SEKAR, V., OGIHARA, M., XU, J., AND ZHANG, H. Data streaming algorithms for estimating entropy of network traffic. In *Proceedings of ACM SIGMETRICS* (June 2006).

[21] LONDON STOCK EXCHANGE PLC. Launch of exchange hosting creates sub-millisecond access to its markets. `http://www.londonstockexchange.com/NR/exeres/04192D02-B949-423D-94E2-683D7506C530.htm`, Sept. 2008.

[22] LU, Y., MONTANARI, A., PRABHAKAR, B., DHARMAPURIKAR, S., AND KABBANI, A. Counter braids: a novel counter architecture for per-flow measurement. In *Proceedings of ACM SIGMETRICS* (June 2008).

[23] MACHIRAJU, S., AND VEITCH, D. A measurement-friendly network (MFN) architecture. In *Proceedings of ACM SIGCOMM Workshop on Internet Network Management* (Sept. 2006).

[24] MAHDAVI, J., PAXSON, V., ADAMS, A., AND MATHIS, M. Creating a scalable architecture for internet measurement. In *Proceedings of INET* (July 1998).

[25] MARTIN, R. Wall street's quest to process data at the speed of light. `http://www.informationweek.com/news/infrastructure/showArticle.jhtml?articleID=199200297`.

[26] MISRA, V., GONG, W.-B., AND TOWSLEY, D. Stochastic differential equation modeling and analysis of tcp windowsize behavior. In *Proceedings of IFIP WG 7.3 Performance* (Nov. 1999).

[27] NGUYEN, H. X., AND THIRAN, P. Network loss inference with second order statistics of end-to-end flows. In *Proceedings of ACM Internet Measurement Conference* (Oct. 2007).

[28] PAPAGIANNAKI, K., MOON, S., FRALEIGH, C., THIRAN, P., TOBAGI, F., AND DIOT, C. Analaysis of measured single-hop delay from an operational backbone network. *IEEE Journal on Selected Areas in Communications 21*, 6 (Aug. 2003).

[29] RAMAKRISHNA, M., FU, E., AND BAHCEKAPILI, E. Efficient hardware hashing functions for high performance computers. *IEEE Transactions on Computers 46*, 12 (Dec. 1997).

[30] RISKA, M., MALIK, D., AND KESSLER, A. Trading flow architecture. Tech. rep., Cisco Systems, Inc. `http://www.cisco.com/en/US/docs/solutions/Verticals/Trading_Floor_Architecture-E.pdf`.

[31] SAVAGE, S. Sting: a TCP-based network measurement tool. In *Proceedings of USENIX Symposium on Internet Technologies and Systems* (Oct. 1999).

[32] SHAIKH, A., AND GREENBERG, A. OSPF monitoring: Architecture, design and deployment experience. In *Proceedings of USENIX NSDI* (Mar. 2004).

[33] SOMMERS, J., BARFORD, P., DUFFIELD, N., AND RON, A. Improving accuracy in end-to-end packet loss measurement. In *Proceedings of ACM SIGCOMM* (Aug. 2005).

[34] SZIGETI, T., AND HATTINGH, C. Quality of service design overview. `http://www.ciscopress.com/articles/article.asp?p=357102&seqNum=2`, Dec. 2004.

[35] TOOMEY, F. Monitoring and analysis of traffic for low-latency trading networks. Tech. rep., Corvil, Ltd., 2008.

[36] VARDI, Y. Network tomography: estimating source-destination traffic intensities from link data. *J. American Statistical Association 91* (1996), 365–377.

[37] WOVEN SYSTEMS, INC. EFX switch series overview. `http://www.wovensystems.com/pdfs/products/Woven_EFX_Series.pdf`, 2008.

[38] ZHANG, Y., ROUGHAN, M., DUFFIELD, N., AND GREENBERG, A. Fast accurate computation of large-scale IP traffic matrices from link loads. In *Proceedings of ACM SIGMETRICS* (June 2003).

[39] ZHAO, Y., CHEN, Y., AND BINDEL, D. Towards unbiased end-to-end network diagnosis. In *Proceedings of ACM SIGCOMM* (Sept. 2006).

[40] ZSEBY, T., ZANDER, S., AND CARLE, G. Evaluation of building blocks for passive one-way-delay measurements. In *Proceedings of Passive and Active Measurement Workshop* (Apr. 2001).