Not All Microseconds are Equal: Fine-Grained Per-Flow Measurements with Reference Latency Interpolation

Myungjin Lee, †Nick Duffield, Ramana Rao Kompella
Purdue University, †AT&T Labs – Research

ABSTRACT

New applications such as algorithmic trading and high-performance computing require extremely low latency (in microseconds). Network operators today lack sufficient fine-grain measurement tools to detect, localize and repair performance anomalies and delay spikes that cause application SLA violations. A recently proposed solution called LDA provides a scalable way to obtain latency, but only provides aggregate measurements. However, debugging application-specific problems requires per-flow measurements, since different flows may exhibit significantly different characteristics even when they are traversing the same link. To enable fine-grained per-flow measurements in routers, we propose a new scalable architecture called reference latency interpolation (RLI) that is based on our observation that packets potentially belonging to different flows that are closely spaced to each other exhibit similar delay properties. In our evaluation using simulations over real traces, we show that RLI achieves a median relative error of 12% and one to two orders of magnitude higher accuracy than previous per-flow measurement solutions with small overhead.

Categories and Subject Descriptors

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

General Terms

Measurement, algorithms

Keywords

Active measurement, approximation

1. INTRODUCTION

Latency is one of the most fundamental properties of packet-switched networks. End-to-end latency directly impacts several critical Internet applications including multimedia applications such as voice-over-IP, video conferencing and online games. While these traditional applications often require end-to-end latencies within 100s of milliseconds, several new types of applications that require extremely low end-to-end latency (in the order of microseconds) have emerged. For instance, high performance computing applications within data center networks [3], storage applications (with industry moving toward Fiber Channel over Ethernet (FCoE) [20]) and, algorithmic trading applications [26] (together constituting multi-billion dollar markets) all require low end-to-end latencies in the order of few microseconds. A small increase in end-to-end latency for trading applications can, for instance, lead to a loss of millions of dollars in lost arbitrage opportunities [26].

To effectively manage low-latency applications, operators require sophisticated tools and techniques for detecting, and more importantly, localizing delay spikes (*i.e.*, finding the router responsible for the high latency) and other performance anomalies in these networks. Once the problem is localized, they can potentially isolate the particular offending flow that is responsible for causing the delay bursts, and reroute the traffic through other paths. In other cases, the operators may upgrade their bottleneck links that are responsible for the underlying delay spikes. Of course, one could argue it may be more important to devise router architectures that guarantee low end-to-end latencies to begin with—indeed, some switches such as those by Arista [9] and Woven [36] provide latency guarantees within 10s of microseconds—in which case, the need for fine-grained measurements is obviated. Unfortunately, anticipating all types of performance problems and application interactions that may occur in a production data center *a priori* is often difficult; fine-grained measurements are therefore still required.

Detecting and localizing latency problems is surprisingly hard today. Routers and switches by themselves offer very little latency measurement and monitoring capabilities; SNMP counters and NetFlow that routers come equipped with are grossly insufficient. SNMP counters provide coarse-grained statistics on a per-port basis, but do not measure latencies. NetFlow provides basic statistics on a per-flow basis such as number of packets and bytes, but *not* latency estimates. ISP network operators monitor the health of their network by injecting active probes to measure end-to-end delays and use tomographic techniques [37, 12] to infer link and hop properties. Unfortunately, for the granularity of measurements required, active probes need to be injected at extremely high probe rate making them not suitable for these low-latency networks. Operators in these networks therefore resort to specialized measurement appliances developed by vendors (*e.g.*, London stock exchange uses those manufactured by Corvil [4]). This type of appliance however tends to be quite costly (around GB£90,000 for a 2×10Gbps box [2]). The high cost of such appliances is a strong inhibitor to ubiquitous deployment across multiple points in the network, or multiple ports of each router, especially within data

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.
Copyright 2010 ACM 978-1-4503-0201-2/10/08 ...$10.00.

center environments that are much more cost conscious than financial markets.

Recognizing these challenges, researchers have recently proposed a new high speed router-level data structure called LDA [23] for measuring delays within routers at high fidelity. LDA addresses the scaling problem of active probes, and cost issue of commercial monitors. While LDA provides a good start, it is by no means sufficient as it is designed to provide aggregate measurements such as average latency across all packets, but *not on a per-flow basis*. Experience indicates that concurrent flows may experience significantly different latencies even when traversing the same given router, and even over relatively short periods of time. Thus, differentiated delay measurements are critical for diagnosing problems, where the aggregate behavior of a router may appear normal, but specific flows and applications may suffer from bad performance.

We illustrate this situation using two motivating examples that are similar in spirit but differ in their context. In the first example, consider a data center provider hosting several different applications, and a particular application experiences bad performance say due to an offending application that is causing periodic bursts of data (referred to as microbursts [6]). In such cases, aggregate statistics such as average latency alone are insufficient, since averages across several million packets may appear normal, while the application is still hurting. A similar issue is the in-cast problem in data centers where synchronized bursts of packets fill switch buffers causing high latencies or even packet loss because data center workloads tend to be barrier-synchronized [35]. While specific solutions may exist for known problems [35], the constant evolution of data centers in scale and diversity may potentially give rise to several unforeseen performance problems. Our second example considers trading networks, where it is possible for financial institutions to obtain specific SLAs from service providers (such as guaranteed less than 100 μs) [1]. In such a context, it is important for the service providers to be able to localize delay spikes and variations that may happen at any of the several hops between the trading party and the stock exchange—diagnosing these customer-specific problems requires not just aggregate, but flow-level measurements.

Having motivated the intuitive need for differentiated measurements, a fundamental question that one may ask is, how much variation exists over several different flows that are simultaneously traversing a given router. In this paper, we explore this question by conducting a measurement study using time-synchronized packet traces collected between two interfaces of a real router, and simulations of backbone traces using traditional queueing models. Our measurement results reveal several fundamental insights: (1) We observe a significant amount of performance diversity among several contemporaneous flows (up to 2-3 orders of magnitude difference). (2) We observe that packets belonging to different flows exhibit significant temporal similarity within short bursts.

We exploit the insights gained from our measurement study to propose a new architecture called reference latency interpolation (RLI) for obtaining per-flow latency measurements in a scalable fashion. Our target is to accurately detect flow latencies in the order of a few 10s to 100 microseconds on a per-flow basis. We wish to detect both average as well as standard deviations of latencies within a given flow. Thus, the contributions of this paper are:

• *Measurement study of performance diversity and the temporal localization of delay*. Using real router traces and simulations, we conduct a measurement study (§2) that reveals our fundamental experimental insight—while concurrent flows can experience diverse performance at longer time scales due to traffic and congestion burstiness, the delay experience by packets from different flow within small localized windows is similar.

• *Architecture for high-fidelity per-flow latency measurements*. Based on the findings in our measurement study, we propose an architecture (§3) that pushes the state-of-the-art in scalable latency estimation solutions beyond aggregate measurements, to provide per-flow latency measurements.

• *Evaluation using real traces and simulations*. We extensively evaluate the efficacy of our architecture (prototype implementation described in §4) using a combination of real traces as well as simulations. In our evaluation, we observe that our RLI architecture achieves a median relative error of 10-12% (§5.1), and up to one to two orders of magnitude lower relative error than existing state-of-the-art schemes under specific configurations (§5.2).

2. DELAY DIVERSITY AND LOCALITY

Before we set out to devise a scalable architecture for fine-grained flow-level latencies within routers, it is important to ascertain that one aggregate latency measure (for which efficient solutions such as LDA [23] have already been proposed) is not sufficient. In this section, we show that there exists significant diversity of performance experienced by concurrent flows traversing the same link, both through qualitative reasoning from the bursty nature of packet arrivals, and through an experimental study. We also observe that the same burstiness properties reduce performance diversity within sufficiently short time intervals; we discuss the ramifications of this observation for the design of a scalable architecture for per-flow performance measurements.

2.1 Flow-level performance diversity

Under many circumstances, flow arrivals are bursty (flows do not commence as a Poisson process) and flow durations are heavy-tailed (as opposed to exponentially distributed) [29]. In such conditions, congestion also tends to be bursty, being concentrated in rarer longer bursts that would be the case for Poisson traffic. Consequently, the performance experience of a flow depends strongly on whether it encounters a congestion burst or not, and the comparative rarity of the bursts means that the normalizing effect of temporal averaging only comes into play for long flows. A further conclusion is that common statistics of delays encountered by a stream of probes (such as their mean or certain quantiles) can vary significantly from those encountered by flows traversing the same link during the same measurement period.

Experimental demonstration. To demonstrate the existence of performance diversity experimentally, we analyzed the packets delays in four datasets described more fully in §4.1: SANJ, CHIC, WEB468 and WEB700. While these are not data center traces, we believe the observations hold true in general. Note that SANJ and CHIC are derived from synthetic queueing times based on a simple FIFO queueing model (more details about the queueing model in §4.2) and real timestamps of packets arrival on an OC-192 interface. Thus we expect it them to provide a realistic representation of the queueing dynamics whose properties underpin our method. In contrast WEB468 and WEB700 are derived from passage of synthetically generated traffic through a real network with real routers, with actual packet arrival and departure timestamps measured at two interfaces of a router. This enables us to capture any effects specific to complexities of actual as opposed to synthetic queueing. Due to space limitations, we will report our results in greatest detail for SANJ and CHIC, more briefly for the others, although *all* confirmed the expected performance diversity.

To study the differentiated delay properties, we classified each packet according to a 2-tuple key comprising the source and destination IP addresses. Table 1 details the broad properties relevant for our study (see Table 2 for other properties). We also capture

Trace	Dur.	2-tuple keys	Pkts/key	Delay mean(ms)	R
SANJ	600s	4.8M	43.9	0.387	27,867
CHIC	600s	4.6M	28.8	0.287	8,052
WEB468	305s	169	15k	0.55	8.7
WEB700	305s	169	23k	3.70	5.5

Table 1: Traces: duration, number of src-dst keys, average number of packets per key, global average delay of all packets, and range factor R of per-key average delay.

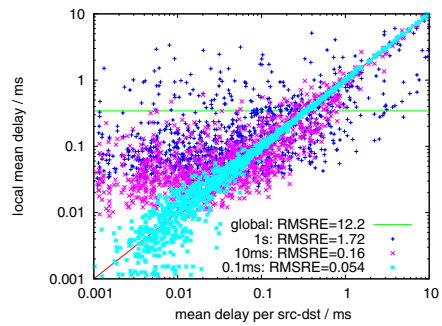
the variability of packet delay across different keys by means of the range factor R described below. WEB700 entails a 50% higher packet rate, as seen from the packet counts over the same duration and correspondingly, the mean delay is about an order of magnitude higher for WEB700. SANJ trace comprises a load about 63% higher than CHIC; mean delay for SANJ, therefore, is about an order of magnitude higher than for CHIC.

In order to further capture the variability of the per key average delay, we calculated the mean delay for each key, and compute the *range factor* R which we define as the ratio of 99th and 1st quantiles of the mean delay per key. We used this measure to capture nearly the full extent of the range, while excluding a small number of outliers. For SANJ, the range spanned three orders of magnitude, for the others about 1 order. We found similar ranges for the 95%ile of the delay within each key in WEB468 and WEB700. We conclude that a single delay statistic, such as an average or quantile over a set of probes over the same duration, cannot accurately account for the delay experienced by the range of traffic flows.

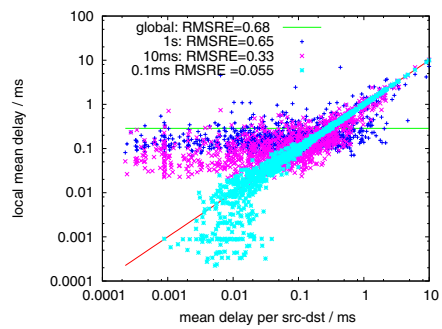
The differences in values of summary statistics between the traces can be interpreted in the light of two factors. First, each keys of WEB468 and WEB700 represents 10s of thousands of packets on average, so there is considerable packet averaging possible. SANJ and CHIC, due to the greater diversity of endpoint addresses, have small numbers of packets per key. However the large range of mean delays seen in SANJ and CHIC are not just an effect brought about by averaging over small numbers of packets. If we restrict attention to the keys with more packets, specifically those with 100 or more (about 85% of the total packets) the range factor is still 36 for SANJ and 18 for CHIC, *i.e.*, still over an order of magnitude.

2.2 Temporal localization of queueing delays

We have just seen how delay statistics of concurrent flows over a 5 minute period can vary over an order of magnitude, and gave a qualitative explanation in terms of bursty nature—both of packet arrivals and congestion. However, this same burstiness additionally leads us to expect that, within bursts of delay, packets should experience more similar queueing delays. A theoretical argument for such behavior has been given in the context of some relatively simple traffic models in [25]. We now demonstrate this empirically, by localizing time, and determining how closely the mean queueing delay experienced by packets of a given flow over small time window can be approximated by the mean delay experienced by the packets of all other flows transmitting packets over the same window. Note that we focus on queueing delay, since different size packets encountering the same delay burst will incur different serialization delays according to their size. Given ingress and egress timestamps, t_i and t_e respectively, of a packet of size b bits at a resource served at service rate r bits per second, the associated queueing delay is taken as $d = t_e - t_i - b/r$. In the remainder of this section, the term "delay" will be understood as queueing delay. We will discuss the ramifications of our findings for the design of performance measurements in §2.4.



(a) SANJ



(b) CHIC

Figure 1: Scatter plot of local vs actual mean delay per source-destination pair; SANJ and CHIC traces, localization intervals 0.1ms, 10ms, and 1s. Global average delay also shown as horizontal line.

In our study, we divide time into fixed interval windows of the same width, and for each key k and interval i, we record the number $n_{i,k}$ of packets present in interval i and their average queueing delay $d_{i,k}$. The average queueing delay encountered by packets during interval i is

$$\tilde{d}_i = \frac{\sum_k n_{i,k} d_{i,k}}{\sum_k n_{i,k}}.$$

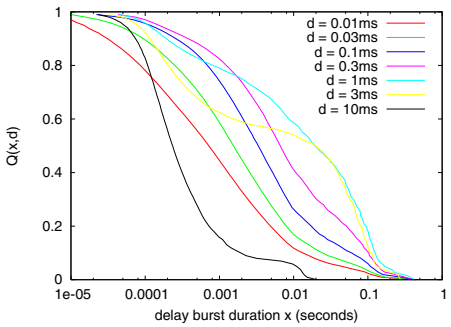
Now the average delay encountered by packets of key k is

$$D_k = \frac{\sum_i n_{i,k} d_{i,k}}{\sum_i n_{i,k}}.$$

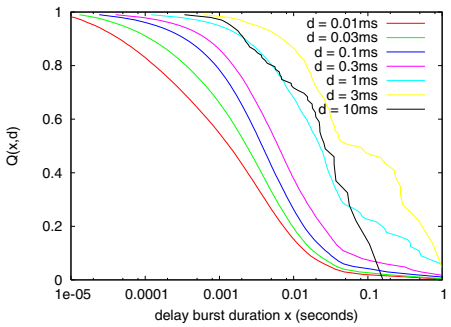
Hence if our intuition is correct, replacing $d_{i,k}$ by $\tilde{d}_i$ in the definition of D_k, *i.e.*, taking a weighted average of the d_i weighted by the numbers of packets $n_{i,k}$ for key k in each intervals, should yield a fairly accurate approximation of d_i, at least for sufficiently narrow intervals. We call the result of the substitution *localized mean delay*, in full it becomes:

$$\tilde{D}_k = \frac{\sum_i n_{i,k} \tilde{d}_i}{\sum_i n_{i,k}} = \frac{\sum_{i,j} n_{i,k} n_{i,j} d_{i,j} / \sum_\ell n_{i,\ell}}{\sum_i n_{i,k}}$$

Figure 1 displays scatter plots of the localized and true mean delays per 2-tuple key for SANJ and CHIC, for localization windows of 0.1ms, 10ms and 1s. For clarity, we show only 1 in 4,000 points. Observe closer agreement for smaller windows, while for large windows the scatter appears to revert to a more horizontal regression, reflecting normalization over longer windows. Note that the localized mean delay is far better predictor of a key's mean delay than the global average packet delay, shown as a horizontal line. We quantify the accuracy via the root mean square relative error



(a) SANJ



(b) CHIC

Figure 2: Delay burst distributions for traces SANJ and CHIC. Proportion $Q(x, d)$ of time spent in bursts of duration at least x, in which delay was at least d, for $d = (0.03, 0.1, 0.3, 1, 3, 10)$ms.

(RMSRE) over all keys, shown in the plot key. Note that even for the smallest localization time (0.1ms), the median number of packets per window was 21 (CHIC) and 35 (SANJ). Thus the accuracy of the localized mean delay is not simply an artifact of comparing a key's packet with itself! Accuracy appears closer for SANJ than CHIC, presumably a consequence of its higher offered load, as evidenced by the longer mean delays in Table 1. We now relate these differences specifically to the burst properties of delay episodes.

2.3 Burst properties of queueing delay

We can further account for the accuracy of the localized mean delay by examining the temporal properties of delay bursts. For this purpose, a burst of delay above d corresponds to a maximal set of some number n of successive packets with arrival times $t_1, \ldots, t_n$ whose delay exceeds d. In this case, the burst duration is taken as $t_{n+1} - t_1$ (or $t_n - t_1$ if packet n is the last packet in the trace). We calculated the proportions $Q(x, d)$ of the time spent a burst of duration at least x in which the delay was at least d.

The displays of Q in Figure 2 account for the differences observed between SANJ and CHIC. As reference delay we take the global mean packet delay δ and ask what duration of bursts the queue spends at least half its time above that level, *i.e.*, what is the duration τ for which $Q(\tau, \delta) = 1/2$? Provided there are sufficiently many background packets in a window of duration τ, we expect the local mean to be fairly accurate. For SANJ, $\delta = 0.39$ms leading to τ of roughly 10ms, while for CHIC, $\delta = 0.29$ms leading to τ of roughly 0.1ms. In both cases this is within the smallest window considered; the larger τ value for SANJ would seem to account for its greater accuracy, apparent in the Figure 1. We also found confirmation of our delay model in relating the burst timescale to the accuracy of localized mean estimates for a given window for the WEB700 and WEB468 traces (omitted for brevity).

2.4 Implications for measurement design

We now tie together the phenomena of performance diversity and delay localization with the problem of per-flow delay estimation. We argue in §3 that a brute force approach in which routers or other devices timestamp every packet is neither necessary nor feasible to produce ubiquitous per-flow delay measurements. The major consequence of performance diversity is that performance statistics of a given flow may differ significantly from those of another (such as a background flow or a probe stream). However, the performance statistics of two sets of packets will agree more closely, if their packets transit at roughly the same times, at least within the typical duration of delay bursts. The crucial observation is that, rather than measuring the delay of each packet in flow directly, it can be sufficient to infer its performance from that of a set of reference packets provided the packet transmission times are sufficiently close.

Now, routers are particularly well placed to create measurements from which to determine the transit delay times of packets. Routers therefore can create a reference stream of packets on a link, giving rise to a reference set of link delay measurements. Then the delay of any given flow can be estimated by selecting measurements from the reference stream that are localized to the packets of the flow under study. This represents a big saving in measurement complexity due to reuse: For different flows, different reference delay measurements are selected as required from the reference stream. Effectively, we can improve the accuracy of the delay measurement of a given flow, by increasing the number of samples contributing to that measurement, specifically selecting those that are most likely to be correlated with it. As an illustrative example, consider a flow with 100 packets. If a sampling rate of 1-in-100 is used, the flow's latency measurements are computed using approximately 1 sample. With our approach, we can compute the latency measurements with all 100 packets, except each packet's latency is estimated using the reference stream (inducing a small amount of approximation error) yielding more accurate results.

In view of the relations between estimation accuracy, delay burst duration, and temporal localization width described above, the approach is contingent on having a probe stream that is sufficiently dense to encounter a typical flows packets within bursts of delays of interest. This is easier to accomplish for high loads, delays being higher and delay bursts being longer. But even when probes are not sufficiently dense for this purpose—resulting in insufficiently narrow localization—we found examples to display no worse accuracy than a naive global average of the type that would be produced by non-local averaging over a probe stream. We remark that a recent approach of leveraging background flow records for delay estimation [25] suffers in this way, because it is inherently unable to control the temporal disposition of reference measurements.

We believe the relevance of these findings for our study is not in the absolute delay values detailed in Table 1, nor the particular localization timescales found in our study. For example, higher speed links may be expected to have shorter queueing delays and hence, shorter timescales for the localization of packet delays. But this effect is compensated for by the fact that a higher packet rate link can be expected to accommodate a higher rate reference packet stream, that can therefore sample delays at a finer granularity. Note also that our findings are more relevant within financial and data center networks, that tend to more stringent in the latency bounds that in a general WAN. For example, a past study [10] found delay jitter across two POPs to be mostly less than 1ms; however, this value can mask the significant diversity amongst smaller delays at the level of microseconds that would still impact performance on a financial network.

3. REFERENCE LATENCY INTERPOLATION

In our setting, we consider a stream of packets traveling from a sender to a receiver (*e.g.*, ingress and egress router interfaces), and we are interested in estimating per-flow latencies. We assume fine-grained time synchronization between the sender and receiver. Within a router, this is straightforward as they both typically operate within the same clock domain. Even across routers, microsecond precision time-synchronization can be achieved with the help of primitives such as IEEE 1588 [15] that are increasingly being deployed within routers. (Note that the error due to clock synchronization is an additive component to the estimates computed by our architecture.) We first quickly discuss possible solutions and see why they may not work well.

3.1 Problems with previous solutions

Naive approach. One way to obtain latency estimates is to maintain timestamps for each packet at the sender and receiver. For estimating per-flow latencies, we just collect the timestamps for all packets that belong to a given flow and aggregate them. The biggest problem with this approach is scale: At 10 Gbps, the number of packets is of the order of a few million per second making it expensive in terms of number of timestamps maintained (memory), of updating timestamps into specific data structures or packets themselves (processing), and transporting the timestamps from sender to the receiver or wherever the latencies are computed (bandwidth).

Packets carrying timestamps: We can potentially embed timestamps within packets, but IP packets currently do not have a timestamp field while TCP options are typically meant for end-to-end latencies. Embedding timestamps require changes to packet headers, and may cause intrusive changes to the router forwarding paths (that often involve third-party components such as TCAMs, switch fabric ASICs) that vendors often refrain from adopting. In addition, adding packet headers to each and every packet can consume significant extra bandwidth that is not desirable. For example, a 32 bit timestamp per packet (assuming minimum size packets of 40 bytes) could use up to 10% capacity.

LDA. If we are only interested in aggregate delay, we could just maintain *two* counters at the sender and the receiver that maintain the number of packets and their timestamp sum. At the end of the interval, the sender could transmit these two counters to the receiver which can subsequently compute the average delay. This is the basic idea exploited in a recently proposed data structure called LDA [23]. In order to account for potential packet loss, LDA uses a stage of sampling and multiple buckets (say 1,000) to ensure that statistics are computed over a large number of samples. While this idea works great for aggregate delays, it is unclear how to extend this idea for obtaining per-flow estimates. The trivial idea of maintaining LDAs with many counters for each and every flow is not likely to scale as the number of flows could be large. Even if we could somehow provision storage for each and every LDA, the sender counters for each flow need to be periodically transmitted to the receivers. Thus, control bandwidth is going to be too high. One could argue that per-flow measurements may be required only for a small subset of "important" flows, in which case, maintaining per-flow LDA (for that subset of flows) would be feasible. Unfortunately, however, it is not often clear which set of flows need to be chosen for per-flow measurements in advance. Besides, determining the right size of the LDA banks may be difficult in advance since flow sizes are not known a priori.

We therefore need to consider alternate mechanisms to achieve our goal. In particular, we can exploit the observations in our previous section (§2) that packets that belong to different flows experience similar delay when they are closely spaced within each other.

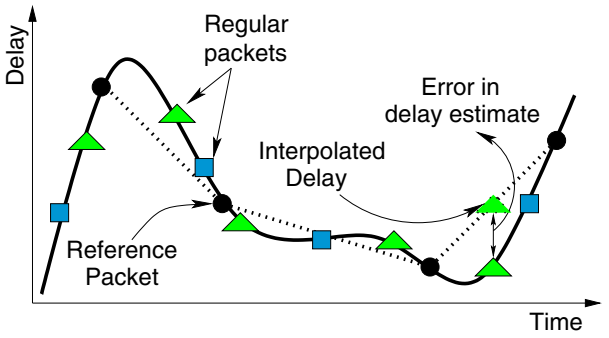


Figure 3: Key idea in our architecture is to estimate packet delays by interpolating the reference packet latencies.

3.2 RLI architecture

Intuitively, queueing delay, which is the major portion of delays experienced in routers, can be thought of as a continuous function (not necessarily monotonic) in busy periods where there are packets to send. In Figure 3, we show the variation of delay as time progresses at the *sender side*. We can observe that the delay experienced by each of the regular packets can be estimated accurately from a few reference delay samples (shown as circles in the figure) by interpolating these reference packet delay samples (shown by a dotted line in the figure). Further, the interpolation error can be controlled by varying the number of reference points in the delay curve, thus trading-off accuracy for resource usage. This is the key idea exploited in our architecture.

Our architecture consists of two main components: a *reference packet generator* at the sender side, and *a latency estimator* at the receiver that maintains a few counters on a per-flow basis. The reference packet generator injects reference packets with sender timestamps periodically into the packet stream at the ingress interface of a router. These reference packets would experience queueing and other effects similar to that of the regular packets thus providing a stream of reference delay samples for the latency estimator at the receiver end. The latency estimator estimates the delay of a regular packet using these reference delay samples that are then accumulated into the per-flow counters.

3.2.1 Reference packet generator

A key question concerns when to generate the reference packet. One option is to inject them according to a Poisson distribution. While in the past, Poisson-modulated probes have been advocated by researchers [28] since they capture time averages very well, our goal is not to compute the average behavior of the queue over a given time. Instead, we wish to use these reference packets to estimate individual packet delays, and thus, Poisson modulation is not a requirement in our system. Furthermore, we wish to bound the interprobe time in order to control the impact of probes on background traffic, whereas Poisson probes can have arbitrarily small interprobe times.

There are three choices we first consider: The first is to inject one reference packet for every n regular data packets (*e.g.*, $n = 1000$). Besides being simple to implement, this 1-in-n reference packet injection has a bounded overhead in terms of number of additional packets injected as a function of the total number of packets. The problem, however, is that there could be periods of low utilization when these reference packet can be spaced apart significantly, potentially affecting the accuracy of the interpolation estimates. To alleviate this, an alternate solution is to inject an active probe packet

Algorithm 1 Reference packet injection rate adaptation

1: **procedure** CALCULATE-INJECTION-RATE
2: ▷ r_{eff}: effective injection rate
3: ▷ d_{rp}: duration between two reference packets (RPs)
4: ▷ c_b: byte counts of regular packets between RPs
5: ▷ u_{est}: moving-averaged link utilization
6: ▷ u_{min}, u_{max}: minimum, maximum link utilization
7: ▷ r_{min}, r_{max}: minimum, maximum injection rate
8: ▷ α: EWMA smoothing factor
9: ▷ l_c: link capacity
10: $u_{instant} \leftarrow c_b/d_{rp}/l_c$
11: $u_{est} \leftarrow u_{instant} \cdot \alpha + u_{est} \cdot (1-\alpha)$
12: $c_b \leftarrow 0$
13: $u_{eff} \leftarrow u_{est}$, where $u_{min} \leq u_{est} \leq u_{max}$
14: $r_{eff} \leftarrow \sqrt{1-(\frac{u_{eff}-u_{min}}{u_{max}-u_{min}})^2}(r_{max}-r_{min})+r_{min}$
15: **return** r_{eff}
16: **end procedure**

every τ time period (*e.g.*, τ = 1ms). While this can result in a fixed worst case bandwidth requirement, this may provide worse results when the utilization is higher, especially when the delay variations are quite rapid. Lastly, we may combine these two approaches by injecting a packet every 1-in-n, or after τ seconds, whichever comes first. Unfortunately, it is not clear how to identify the right value of τ. On one hand, keeping τ low increases accuracy but causes too much overhead and starts to interfere with regular packets. On the other hand, setting a high value of τ defeats the purpose of setting an upper bound on the time-period.

Thus, we consider monitoring the utilization in a dynamic fashion in order to determine at what time instants to inject the packets. We find that this *adaptive* scheme performs better than either of the fixed time based or count based schemes just described. Adapting the probe rate to utilization enables us to get the best of both worlds: limiting the probe rate at high utilizations, while getting sufficiently frequent coverage at low utilizations. Still, adaptive schemes entail a subtle trade-off because the adapter may lag in response to a high rate burst of shorter duration than its adaptation timescale. In practice, however, we have not found such phenomena to degrade the performance experienced by background traffic.

Although we expect the advantage of adaptation to be generic, we now discuss the particular form of realization in our implementation. To keep track of link utilization and adjust reference packet rate, we maintain small amount of state. Specifically, our adaptive scheme consists of two steps: updating link utilization and calculating effective reference packet rate. Algorithm 1 presents the pseudocode for the scheme. The algorithm is triggered right after a reference packet is injected with the previously calculated effective reference packet rate r_{eff}.

To estimate link utilization, we maintain a byte counter c_b that keeps track of the number of regular packets between two injected reference packets. We also maintain the time interval d_{rp} between the two injected reference packets. We calculate instantaneous link utilization using these two variables and link capacity l_c. We could use the instantaneous link utilization $u_{instant}$ directly to calculate effective reference packet rate, but, in order to remove the effects of short term fluctuations of estimated link utilization, we update average link utilization u_{est} using exponentially weighted moving average (EWMA) with a smoothing factor, α. We reset the byte counter immediately after link utilization estimation is done.

After updating u_{est}, we calculate the next reference packet rate. Our objective is to adapt r_{eff} as a function f of link utilization,

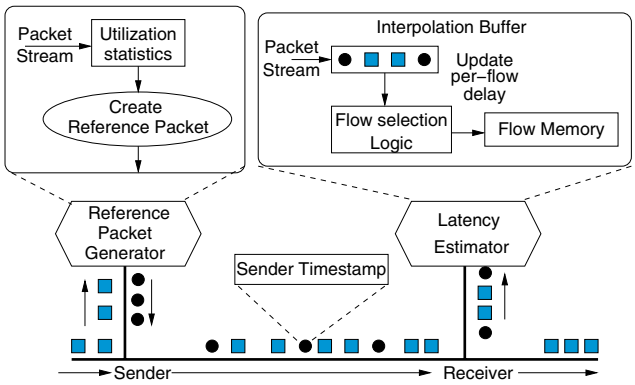


Figure 4: Overview of our architecture.

where $r_{max} = f(u_{min})$ and $r_{min} = f(u_{max})$, u_{min} and u_{max} being configurable parameters. Thus, we bound u_{eff} to ensure that u_{eff} always lies in between u_{min} and u_{max}. If u_{est} is higher (lower) than u_{max} (u_{min}), we set u_{eff} is set to u_{max} (u_{min}). While there could be many choices for the function f, we choose an elliptical function (shown in line 14) and calculate r_{eff}. The rationale for choosing this function is that, it typically targets accurate estimation of latency under low to moderate utilization (*i.e.*, decreases r_{eff} slow when u_{eff} is close to u_{min}), but reduces rate significantly at high utilization (as u_{eff} approaches u_{max}). For our evaluation, we set u_{min} = 0.6 and u_{max} = 0.85, while r_{min} and r_{max} are set to 1-in-300 (0.0034) and 1-in-10 (0.1) respectively.

3.2.2 Latency estimator

The receiver processes the reference packets (containing timestamps) inter-mixed with regular data packets to estimate per-packet latencies. Our architecture does not require the receiver to maintain counters for all flows in the network. Indeed, our architecture can work on top of any existing framework for per-flow measurements such as NetFlow, that maintain flow records (containing number of packets, bytes, etc.) for a small subset of flows. For each of the flows of interest (obtained using any flow sampling schemes), we maintain three counters indexed by the flow key that keep track of the following: (1) number of delay samples for the flow; (2) sum of estimated delays for all packets of that flow; (3) sum of squares of individual packet delays. This composite set of counters are updated for *all* packets that belong to flows of interest. It is, therefore, important to implement these counters in high-speed SRAM to scale to high line rates. (We discuss other alternatives later in §6.)

Our latency estimator component also contains an *interpolation buffer* (as shown in Figure 4) to store packets that have arrived between two reference packets. This requirement stems from the fact that delay value estimated for each individual packet is a function of the delay experienced by the two reference delay samples (corresponding to the reference packets). Of course, we do not need to store the entire packet in the interpolation buffer; storing just the flow key, the associated timestamp and byte count are sufficient for each packet. The size of the interpolation buffer required can be statically determined depending on the design of the reference packet generator. If reference packets are generated according to the 1-in-n scheme, the interpolation buffer need not be larger than n. For other schemes, we can easily compute an upper bound on the number of packets between two active probes. For instance, for the 1-in-τ scheme, we can easily compute the number of minimum-size packets for a given link capacity that can be transmitted in τ seconds; this dictates the size of the interpolation buffer.

While the presence of the interpolation buffer in our architecture facilitates the use of both left and right reference packets to estimate delay for a given packet (potentially allowing better accuracy), it requires additional complexity in state maintenance. At the other end of the trade-off, we can imagine getting rid of the buffer completely and estimate the delay of a packet as a function of only the reference packet before the packet, but *not* after. This requires no state in terms of the interpolation buffer, but requires remembering the delay experienced by the reference packet, that can be easily kept track of using a single counter.

3.3 Packet delay estimators

We formally describe our packet delay estimators in this section. The first estimator called RLI estimator[1] utilizes two reference packets for linear interpolation and works as follows.

RLI estimator. Let p_i^a be an i-th reference packet. Let $p_j^r,\ j = 1, 2, \ldots, n$ be a regular packet whose receiver timestamp is located between $a_l = p_i^a$ and $a_r = p_{i+1}^a$ that represent the left and right reference packets in the interpolation buffer. Let τ_j^r and b_j^r denote the receiver-side timestamp and a byte count of p_j^r, and τ_l, τ_r represent the receiver-side timestamps of a_l and a_r. Let b be the size of reference packet and l_c be the link capacity. Then the estimated delay, $\hat{d}_j$ for p_j^r obtained by interpolating the delays of a_l and a_r (represented as d_l and d_r) is given as:

$$\hat{d}_j = d_l + (\tau_j^r - \tau_l)\frac{d_r - d_l}{\tau_r - \tau_l} + \frac{b_j^r - b}{l_c},\ j = 1, 2, \ldots \quad (1)$$

The third term on the right-hand side in Equation (1) compensates for different serialization times by the difference in packet size between regular packets and reference packet. Whenever a new probe packet arrives, a_l and a_r are updated; subsequent interpolated delays of new regular packets are computed with these new values as given by Equation (1).

For each flow f_k, three per-flow counters are maintained as we discussed before. After the delay estimate is computed for the packet p_j^r, the counters corresponding to the flow to which p_j^r belongs are updated as follows.

$$c(f_k) = c(f_k) + 1 \quad (2)$$

$$m(f_k) = m(f_k) + \hat{d}_j \quad (3)$$

$$v(f_k) = v(f_k) + \hat{d}_j^{\,2} \quad (4)$$

When a flow with a flow key f_k expires, if $\widetilde{m}(f_k)$, $\widetilde{v}(f_k)$, and $\widetilde{c}(f_k)$ represent the final values of the number of, mean and variance counters in flow memory, then the delay mean and variance of a flow f_k are:

$$\mathsf{E}[d_{f_k}] = \widetilde{m}(f_k)/\widetilde{c}(f_k) \quad (5)$$

$$\mathsf{Var}[d_{f_k}] = \widetilde{v}(f_k)/\widetilde{c}(f_k)^2 - \mathsf{E}[d_{f_k}]^2 \quad (6)$$

where d_{f_k} denotes a random variable for delays of packets of a flow with f_k. These values are updated before exporting the flow record.

RLI-L estimator. The RLI estimator requires storing packets in an interpolation buffer until a reference packet arrives after which each of the packets' delays are updated, that requires additional complexity. Thus, we consider an alternative estimator called RLI-L estimator that instead of using both the left and right delay samples uses only the left delay estimate. In other words, for all regular packets that appear between p_i^a and p_{i+1}^a with delays d_l and d_r,

$$\hat{d}_j = d_l + (b_j^r - b)/l_c.$$

[1]We use RLI estimator to refer to the estimator and just RLI to refer to the architecture.

Name	#flows	#packets	pkts/flow	Date
Link: OC-192 (10 Gbps), Duration: 600s.				
CHIC	4.56M	131.42M	28.8	Apr. 30 17:00:00 2008 UTC
SANJ	4.87M	213.80M	43.9	Dec. 18 05:55:00 2008 UTC
Link: OC-3 (155 Mbps), Duration: 305s.				
WEB468	0.143M	2.61M	18.25	Feb 6 02:06:58 2006 UTC
WEB700	0.214M	3.99M	18.65	Feb 6 02:15:32 2006 UTC

Table 2: Trace characteristics: CHIC and SANJ are OC-192 ISP traces. The other two are real router traces with synthetic traffic (jointly referred to as WISC).

The per-flow counters are updated the same way as before in RLI estimator. Because this estimator does not use both values, it is not as accurate as the RLI estimator as we shall discuss in our evaluation.

Shrinkage estimation. While linear interpolation is a simple means to approximate the delay, linearity may not always be the best choice. We therefore considered possible refinement of the delay estimators, in particular using Shrinkage Estimation [11]. This is a standard method to improve accuracy of an estimator X by making a convex combination of it with some fixed point X_0, possibly 0, to form $X' = \lambda X + (1 - \lambda)X_0$ for some $\lambda \in [0, 1]$. In our case, the approach is to shrink individual delay values towards an EWMA estimate of the mean delay. It turned out that Shrinkage Estimation provided only a very small improvement in estimation accuracy in the results reported in §4. We consider exploring other non-linear interpolation schemes as part of our future work.

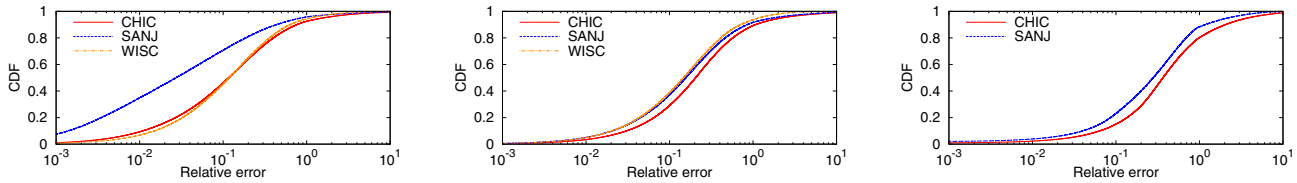
4. EVALUATION METHODOLOGY

We evaluate our architecture in order to answer the following questions. (1) How accurate is our architecture in estimating per-flow latencies under different settings? (2) How does our architecture compare with previous solutions such as trajectory sampling [14] and Multiflow estimator [25]? (3) What are the overheads involved in our measurement architecture? (4) Does the architecture cause any interference with regular packets? In order to answer these questions, we build a custom simulator that uses packet header traces as input (using either synthetic or anonymized IP addresses), and implements the reference packet generator and latency estimator modules to simulate the architecture.

4.1 Data sets used for evaluation

Ideally, we need real packet traces with high-resolution timestamps of ingress and egress interfaces of a production router. Unfortunately, we have found no such public datasets, neither in the target setting of a data center or trading network, nor otherwise. The only known study to us, although in a backbone ISP context, is the one by Papagiannaki *et al.* where they conducted extensive studies of delays in real routers by collecting GPS synchronized packet header traces from the Sprint network [27, 19], but the data set itself is not public. In their study, the authors observe that packet delays follow a Weibull distribution. However, our method exploits the correlations of the delay across different packets, which are not modeled in their study.

In the absence of ideal traces, we resort to two other types of traces. First, we used traces of the passage of synthetic traffic across a real router collected by the authors of [33] in their evaluation of a new active probing tool. Traffic is synthetically generated using the Harpoon traffic generator [32] over a dumbbell topology with an OC-3 bottleneck link. Even though the traffic sources are synthetic, they are subject to *real* router forwarding paths, queueing and other behavior, and thus are quite realistic. The router employed the RED queueing policy during collection of these traces.



(a) High utilization (≈ 88%) (b) Moderate utilization (≈ 55%) (c) Low utilization (≈ 22%)

Figure 5: CDF of mean per-flow delay estimates using RLI estimator for different utilizations and different traces.

Ingress and egress timestamps are recorded for each packet, hence the delay incurred in traversing the router is computed by subtraction. The data set referred to as WISC consists of two traces with different utilization levels, summarized in Table 2.

Second, we used backbone header traces published by CAIDA [31] that include actual packet arrival times of real packets at an interface, and then simulate the passage of these packet arrivals through a queue. The traces are summarized in Table 2. Each trace records packet arrivals during a 600 second period on OC-192 (*i.e.*, 10 Gbps) backbone links of a tier-one ISP. The IP addresses in the trace were anonymized. The traces denoted as CHIC and SANJ represent those collected at Chicago, IL and San Jose, CA respectively. For the experiments described below, we classified packets according to the standard 5-tuple comprising source and destination IP addresses, port numbers and protocol fields. (Note the distinction with the 2-tuple keys employed in §2.)

4.2 Simulator

While the ideal prototype implementation would use a real router, router architectures are typically proprietary making it hard to make any changes. Further, to demonstrate the effectiveness of our architecture, we mainly need sender and receiver timestamps for individual packets that pass through a real router; the internal details of routers are not important for our evaluation. Thus, we build a simulator by extending an open-source NetFlow platform called YAF [8] for our simulation. NetFlow, the de facto passive measurement solution, already supports flow-level collection of basic statistics such as number of packets, bytes, etc. Thus, extending YAF automatically provided us with the flow creation, flow update, and flow expiry mechanisms in regular NetFlow. We added support for the injection of reference packets from the sender side, the interpolation buffer at the receiver, and latency estimator along with three additional counters we maintain for the latency estimates on a per-flow basis. We implement the adaptive reference packet injection algorithm based on keeping track of the utilization as described in Algorithm 1. Since most real routers use RED, we simulate RED queue management strategy.

In the queueing model employed for simulation with CHIC and SANJ, we control the packet loss and delay by configuring queue length and drain rate. We fix the drain rate in terms of bytes per second. By fixing this one parameter, the drain rate, we can automatically control both the delay as well as the loss distribution. Following the guidelines in [18], we chose a queue size of 10,000, $min_{th} = 4,000$ and $max_{th} = 9,000$, queue weight $w_q = 0.002$ and maximum drop probability, $max_p = \frac{1}{50}$ for all traces. Note that while our simulation is open-loop, *i.e.*, we do not see TCP backoff effects even when we drop packets using RED, both WEB468 and WEB700 traces are generated by configuring a real router with RED, and as such, will expose all the relevant TCP backoff dynamics associated with RED.

For WISC traces, since we cannot easily inject reference packets into the simulation, we rely on a simple packet marking scheme that denotes the nearest regular packet as a reference packet whenever it needs to be injected. Compared to adjusting the packet timestamps to simulate the injection of a reference packets, our packet marking scheme is much less intrusive. We believe that it does not affect the accuracy of our architecture, because the delays are still real packet delays. Effectively, the reference packet times are just slightly offset from what they would be in an actual realization.

4.3 Other solutions for comparison

Trajectory sampling. First, we consider trajectory sampling proposed by Duffield *et al.* to sample packet trajectories [14]. While the original intent is different, we can add a timestamp with each packet label sampled at a router, and aggregate samples that belong to a given flow for latency estimates. The estimator just computes the difference of timestamps at two adjacent locations (similar to the naive timestamp idea discussed in §3).

Multiflow estimator. Second, we consider a new estimator called Multiflow estimator (MFE) proposed by Lee *et al.* in [25]. MFE exploits the fact that NetFlow already maintains timestamps of start and end packets for each (sampled) flow. Two adjacent routers using consistent hash-based sampling will collect same flow records with same start and end packets, giving two delay samples. Given that the simple averaging of just these two samples is not an accurate estimator, MFE computes the average of all delay samples (referred to as background samples) that may potentially belong to other flows within the start and end of the flow. The spirit behind this estimator is grounded in a similar observation as ours in §2.

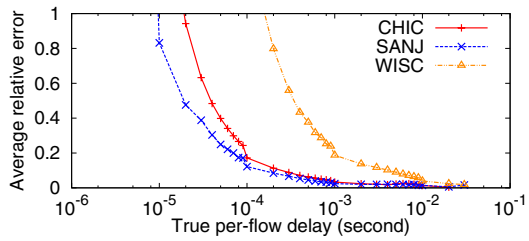
5. RESULTS

We divide our results into three major parts: First, we evaluate the accuracy of our RLI estimator, both mean and standard deviation estimates, for different traces and different utilizations. Second, we compare our architecture with other solutions such as the trajectory sampling and MFE described in §4.3. Finally, we evaluate the overheads involved in our architecture.

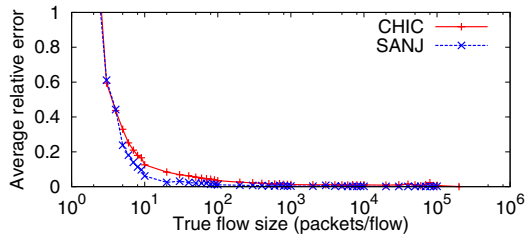
5.1 Accuracy of RLI

To evaluate the accuracy of RLI, we primarily focus on the relative error (defined as $|true - estimated|/true$) of mean and standard deviation estimations of each flow with the ground truth.

Accuracy of mean latency. We plot the cumulative distribution function (CDF) of the relative error of mean delay estimates for all the flows in Figure 5 for different utilizations and traces. In our evaluation, we consider the WEB468 as a moderate utilization scenario with about 55% link utilization, while WEB700 comprises the high utilization scenario (about 88% utilization). We do not have access to a lower utilization trace in the WISC data set, hence we do not show the curve for WISC in Figure 5(c). For high and moderate utilizations, we can observe that median relative error of latency estimates among all flows is around 10-12%. The 75%ile relative error is also less than 20% in these two cases. For low utilization, median relative error of estimates is around 30%. Across

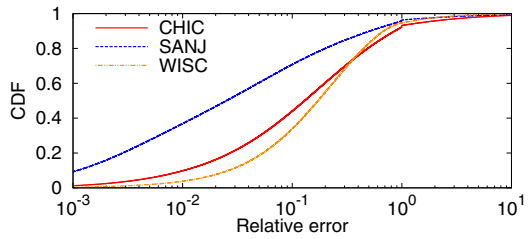


(a) Binned by delay

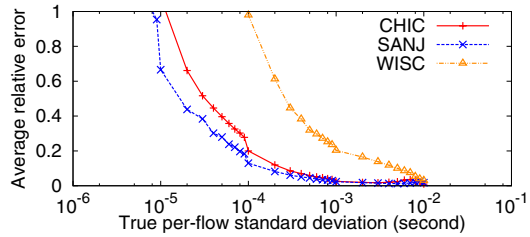


(b) Binned by flow size

Figure 6: Average relative error of per-flow delay mean estimates binned by true flow delays and sizes.



(a) CDF



(b) Binned by standard deviation

Figure 7: Average relative error of per-flow delay standard deviation estimates binned by true deviations.

different curves, we observe that the accuracy is largely similar both for real router packet traces (WISC) as well as our backbone traces (CHIC and SANJ traces).

In general, we observe that the accuracy of RLI appears better for high utilization than low utilization cases. To put this result in perspective, we observe that true value of the median delay directly depends on the utilization levels (under high, moderate and low utilizations, average per-flow latencies are approximately 390, 11.4, 1.0μs for CHIC trace). Recall that the utilization levels in our simulations are governed by the queue drain rate; if the drain rate is low (high), utilization is going to be low (high). Low utilization automatically translates into low per-packet queueing delay for most packets, and hence, most flows. In other words, serialization time is a more dominant factor in latency than queueing delay. Further, serialization time varies depending on packet size, which in turn causes more jitter in interpolation process; errors in low utilization are, therefore, more significant than in high utilization.

We envision that our architecture is more suitable for isolating the router where a flow experienced bad end-to-end performance; we therefore care about flows for which the delays are significantly higher than the rest. In other words, for small delays (*e.g.*, 10μs), a relative error of 100% is not nearly as significant as compared to flows which experience higher delays (*e.g.*, 100μs). Thus, while one could argue that network operators may operate typically at low utilizations, the accuracy of RLI is itself oblivious to the exact utilizations, and mainly depends on the absolute latency of a given flow. The fact that the accuracy of our architecture appears significantly better in the higher utilization case is merely a reflection of the fact that the number of high latency flows is higher in this case. (This also explains our rationale in designing our adaptive reference packet injection strategy to reduce the rate as utilization grows significantly.) In order to bring this out in more detail, we group flows by delays and flow sizes.

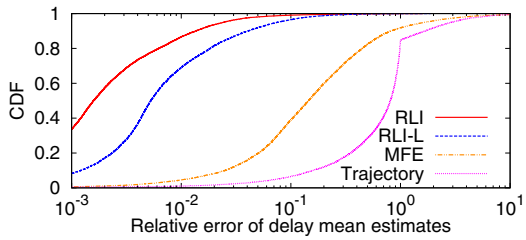
Grouping flows by delays. In Figure 6(a), we plot average relative error of delay mean estimates by grouping relative errors based on true per-flow delay. In the figure, we only plot high utilization condition, because 99.99% of per-flow latencies found in both moderate and low utilization scenarios are quite low for both traces (at most 90μs and 10μs respectively). Since we are more interested in the high delay flows, for brevity, we mainly focus on the top 50% that start at an average latency of about 100μs all the way until about 30ms.

We can observe from Figure 6 that RLI is quite accurate in measuring latencies of flows that exhibit large delays. Average relative error of mean delay estimates is close to 12% for flows with true delay greater than 100μs (in the SANJ trace). For the CHIC trace, we found that 75% of flows having about 100μs latency have less than 18% relative error in CHIC trace, slightly higher than the SANJ trace. Of course, relative errors typically go down as the true delay increases as the denominator is getting bigger. The important thing, however, is that absolute error is not growing proportionately and remains relatively small and bounded; thus, our solution can be quite effective in measuring flow-specific delay spikes of the order of a few 100μs very efficiently—exactly the level of SLA specifications that Cisco provides in its trading floor architecture [1].

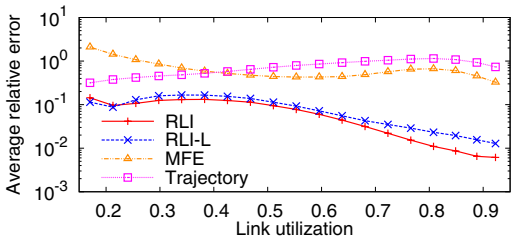
Our real router trace, WISC, shows similar trends with CHIC and SANJ in that as true delay increase, average relative error decreases significantly. Specifically, for top 20% of delays which is around 3ms true delays, RLI achieves less than 11% average relative errors. Recall that the WISC trace is collected over an OC-3 link, that is 64 times less capacity than the OC-192 backbone traces. Thus, intuitively, 100μs delay in the OC-192 trace translates loosely to around 6.4ms in the WISC trace, for which the error in the delay is around 8-9%, similar to the backbone traces.

Grouping by flow sizes. As we have considered flows with large delays before, operators may also care more for larger flows, for which latency effects may be more pronounced than smaller ones (say with fewer than 10 packets). Thus, in Figure 6(b), we plot the average relative error for flows binned by their sizes. In our results, we found that the top 20% of flows had more than 10 packets in our backbone traces—average relative error for these is less than 11%. For larger flows, the error is even lower (around 3% for flows larger than 100 packets). Flow-size distributions in WISC traces are synthetic; hence, we did not plot the corresponding curve.

Accuracy of delay standard deviation. While good accuracy in average latency estimates is nice, it is important to be able to esti-



(a) CDF of relative error



(b) Average relative error by utilization

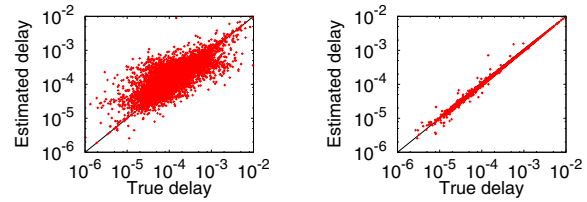
Figure 8: Comparison with other solutions on SANJ trace with a packet sampling rate of 0.1%.

mate the variation in delays accurately, at least for the flows where such a measure is important, *i.e.*, those that exhibit high amount of standard deviation. We follow a similar approach as we did for mean delay to compute a CDF of the standard deviation estimates (shown in Figure 7(a)). From the figure, we can observe that the median error is less than 12% with some small fraction of flows exhibiting high relative error. We also computed similar CDFs for the moderate and low utilization cases (not shown for brevity). As in the case of mean, the standard deviation estimates were more accurate for the high utilization case as compared to the low utilization scenario. In all utilization cases, when the true value of the standard deviation is quite low, we found that the relative error was really high—the exact proportion of flows that exhibited low standard deviation changed depending on the utilization characteristics. To show this, we bin the flows into different groups based on their true standard deviation and plot the average relative error in Figure 7(b). As before, we can observe that the average relative error in detecting standard deviations greater than $100\mu s$ is less than 20%, and for higher standard deviations, it is even lesser.

5.2 Comparison with other solutions

We compare our architecture with previously proposed solutions discussed in §4.3, trajectory sampling and MFE, and also study the advantage of using RLI estimator compared to RLI-L estimator. In these experiments, we introduce a sampling rate of 0.1% in order to keep the trajectory sampling overheads relatively small. While the MFE and RLI estimator (both variants) do not care about the packet sampling rate directly, it affects the set of flows created; typically, random packet sampling leads to the creation of flow records for relatively large flows or 'elephants'. Thus, to make the accuracy comparison consistent (on the same set of flows), we subject our RLI to the same sampling rate as both trajectory and MFE. Note that for RLI, we estimate and update the packet latency counters for all the packets (similar to sample-and-hold [17]) that match the flow *after* the flow is created.

We plot the CDFs comparing the relative errors of mean delay estimates (standard deviation graphs look very similar) across different schemes in terms of relative error in Figure 8(a) for the high



(a) MFE (b) RLI

Figure 9: Scatter plots comparing the flow latency estimates output by MFE with RLI.

utilization case. Trajectory sampling clearly performs the worst, in part because it contains very few samples on a per-flow basis. The relative error for about 50% of the flows is larger than 80%; the estimates therefore are not reliable at all. MFE performs better than trajectory; the fact that it takes advantage of intermediate background samples from other flows into consideration allows it to refine its estimates, allows it to approach the global mean observed during the duration of that particular flow.

RLI performs the best among all with most estimates well within 1% relative error—representing two and half orders of magnitude improvement over trajectory sampling (500x) and almost two orders improvement over MFE. We observe a similar trend with the standard deviation estimates (not shown in the figure). RLI-L, that uses no interpolation buffer and assigns the delay observed because of the left reference packet as the estimate, performs better than both MFE and trajectory, but loses some amount of accuracy (about half an order of magnitude) compared to RLI. This is the price RLI pays in the form of an interpolation buffer to hold packets.

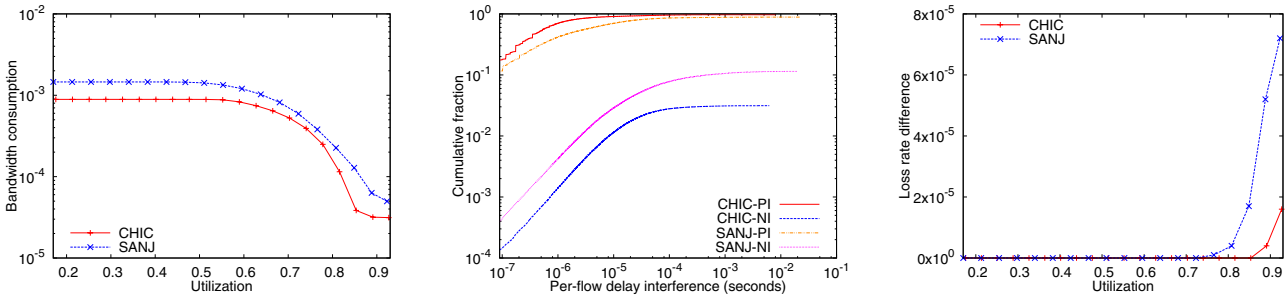
We also compare these solutions across a wide range of utilizations in Figure 8(b). We observed similar trends, with RLI outperforming the rest significantly; the gap between RLI and the rest is significantly more pronounced at higher utilizations where the absolute delays tend to be high for which the accuracy of RLI is much better. Still, RLI estimates are more than an order of magnitude more accurate than MFE in many cases.

Scatter plots in Figure 9 compare the true and estimated delays for individual flows for both RLI and MFE. The key difference between RLI and MFE is the presence of extra timing information for each and every intermediate packet that belongs to that particular flow. This timing helps RLI estimate the delay significantly better using a local estimate (as discussed in §2) compared to the MFE scenario, where it crudely approximates based on the sampled first and last packets of the background flows. The difference can be clearly observed in all our results, with RLI outperforming MFE by one to two orders of magnitude in terms of the relative error.

5.3 Overheads of RLI

We quantify the direct and obvious overheads associated with the reference packet traffic, and indirect effects of the reference packet traffic on actual per-flow latencies and losses. In Figure 10(a), we show the fraction of link capacity used by the reference packet traffic for different link utilization levels. As we can observe, the bandwidth consumed by reference packet traffic is quite small. At low utilization, where the reference packet traffic is injected at relatively higher rate (at roughly 1-in-10 packets), still, the overall bandwidth consumed is about 0.1%. As utilization increases, the bandwidth consumption falls down steeply to 0.007% at 90% utilization.

Low bandwidth is nice, but it is also important for the reference packet traffic to not interfere too much with regular traffic, although some amount of interference is unavoidable. To quantify this, we measure the difference between the average latency experienced



(a) Bandwidth overhead (b) Interference with regular traffic (c) Impact to packet losses

Figure 10: Quantifying overheads in RLI and the interference of reference packet packets on regular flows.

by a flow with and without our architecture. In Figure 10(b), we show the cumulative fraction of flows which experienced a particular amount of additional delay for the high utilization case, where interference is (expectedly) the most predominant. It is natural to expect that flows will potentially experience a positive additional delay (curves x-PI, PI means positive interference). However, we also found about 10% flows for which the average delay went down (curves x-NI, NI is negative interference).

From the distribution of both these types of flows (PI and NI), we observe that there exist flows that appear to have a significantly high (by 20-30ms in one case!) as well significantly low (by almost the same amount) delay in the presence of reference packet traffic compared to when they were absent. Further investigation revealed a significant difference in the number of packets that were dropped; for flows with reduced packet delay, a lot of packets that were not dropped before were getting dropped in the presence of reference packet traffic. Similarly, for flows that experienced increase in delay, we observed the opposite phenomenon, where packets that were getting dropped before somehow survived, although with a huge delay. Put differently, both these incidents—a packet getting dropped or getting delayed significantly—are really related to how close to being full the queue is. Reference packets cause a small perturbance to only the packets at the fringe, with some dropped packets getting converted to high delay and vice-versa; thus, the interference is therefore quite minimal.

In terms of overall loss, we find that our architecture introduces very little increase in the loss rate as can be seen in the Figure 10(c). On the whole, we can find that the packet loss rate differs by at most 0.001% even at almost 80% utilization for either traces. SANJ trace experienced slightly more losses than the other because the arrivals are a little bit more bursty in that trace.

6. IMPLEMENTATION

We now outline how our architecture can be implemented in routers. While the packet generator component itself can be implemented in software as the reference packet rate is not too high, it needs a precise timestamp at the sender side for which hardware is preferable. Our adaptive reference packet generation scheme maintains a little bit of state in the form of a few utilization counters that can be accommodated within the line-card ASICs. While the reference packets need not be routable (IP headers are not necessary), they need to be transmitted from each interface to every other interface and therefore need interface identifiers. (Typical routers maintain a small internal header anyway that can be overloaded for this purpose.) Because the two interfaces are operating within the same time-domain we may not need extra time synchronization (*e.g.*, using GPS clocks). On the receiver side, we mainly require three hardware counters on a per-flow basis for flows of interest. Given the high line rates, counters need to be in SRAM. One can also leverage the hybrid SRAM-DRAM architecture commonly used in managing counters [30] and packet buffers [21], to ensure that high speed counter updates happen in SRAM that are flushed periodically to cheaper DRAMs. Another solution is to report per-flow measurements only for a subset of flows, by sampling or with the help of other mechanisms (*e.g.*, ProgME [38]).

We could potentially compute the per-flow measurements by mirroring all the packets from the receiver to a PC (using span ports available on routers and switches or, using OpenFlow [7]), or to a network appliance capable of processing them (dedicated for these fine-grained measurements). Of course, since the line rates are high, we still need the counter-update functionality within hardware, for which we could leverage FPGA-based solutions (*e.g.*, NetFPGA [5] handles 4 Gbps), or high-speed network processor boards (*e.g.*, Intel IXP 2800). The main advantage of these solutions is that they provide an easy path to deployment today, without having to wait for router vendors to adopt the architecture to facilitate per-flow measurements, although, we believe, that would be the ultimately preferable. A big win for our architecture compared to commercial boxes such as those by Corvil [4] is that we do not require both senders and receivers to obtain measurements; our solution allows obtaining measurements directly at the receiver.

7. RELATED WORK

While designing router-based passive measurement solutions is a well-established area of research, designing solutions for fine-grain latency estimation is a relatively new line of research. This is in part due to the relatively recent advent of applications such as data center and algorithmic trading applications that demand sophisticated tools to assist in debugging and troubleshooting their performance.

There exists a lot of literature (*e.g.*, [37, 12, 39]) on tomography techniques to infer hop and link characteristics from end-to-end measurements (conducted using tools such as [34]) and topology information. They provide aggregate measurements, but not on a per-flow basis. In this context of flow measurement, there have been a wide variety of solutions proposed (*e.g.*, [38, 17, 16, 22]) that employ sampling to control flow selection. Our latency measurement approach proposed in this paper should, for the most part, work seamlessly in many of the sampling frameworks; we have shown how our results compare with other solutions in the context of random packet sampling used by sampled NetFlow.

Duffield *et al.* proposed trajectory sampling in [14], and using them for inferring loss and delay in [40, 13], which we have discussed before. Lee *et al.* [25] exploit the two timestamps already stored on a per-flow basis within NetFlow to obtain a crude estimator called Multiflow estimator for per-flow latency estimates. Since it is related to our goal, we compare the efficacy of this solution

with our architecture in our evaluation. Researchers have in the past conducted measurement studies to understand single-hop delays [27, 19] and delays across PoPs in [10]. They do not propose any architecture for measuring per-flow delays however.

A very recent and one of the most relevant efforts is the design of a data structure called LDA by Kompella *et al.* in [23] and an architecture that encompasses LDA in [24]. While LDA enables fine-grained network latency measurements in the context of new and emerging class of data center and algorithmic trading networks, it only provides aggregate measurements that, as we argued in this paper, is not sufficient since concurrent flows tend to exhibit significant performance diversity. It would be an interesting future direction to design new data structures more similar to LDAs that can achieve per-flow measurements in a scalable fashion.

8. CONCLUSION

Many new applications such as algorithmic trading and data center applications demand low end-to-end latency in the order of microseconds. While special purpose measurement devices can help network operators detect latency spikes and other performance anomalies, the high cost of such devices is a strong inhibitor to ubiquitous deployment. A recent solution, LDA, provides a scalable and low-cost alternative to delay measurement, but only for aggregate traffic, and is hence insufficient for isolating problems that affect specific applications or flows.

We propose a scalable architecture called RLI for obtaining per-flow latency measurements across interfaces within routers. Our architecture is based on two key ideas. First, packets within a given burst encounter similar queueing and other behavior and hence, exhibiting similar delays. Thus, we inject periodic reference packets at the sender with a timestamp that the receiver can use as a reference latency sample. Second, the delay experienced by packets that arrive between multiple reference packets can be approximated by linearly interpolating the delays of the two reference packets. Using simulations on packet traces, we find that RLI achieves a median relative error of 10-12% (§5.1), and one-two orders of magnitude lower relative error compared to previous solutions (§5.2). Another big win for RLI comes from the fact that measurements are obtained directly at the receiver without the need for sender-side packet timestamps for all the regular data packets, in contrast to solutions that require correlating large numbers of packet timestamps collected from multiple points. Our architecture is simple to implement and is cost effective making it practical for ubiquitous deployment. We believe that it offers a compelling alternative to high-end expensive monitoring boxes for network operators.

Acknowledgments

The authors are indebted to the anonymous reviewers and Sue Moon, our shepherd, for comments on previous versions of this manuscript. We also thank Joel Sommers for providing us with real router traces. This work was supported in part by NSF Award CNS 0831647 and a grant from Cisco Systems.

9. REFERENCES

[1] Cisco trading floor architecture. http://www.ciscocapital.info/en/US/docs/solutions/Verticals/Trading_Floor_Architecture-E.html.

[2] Corvil tool minimises latency. http://www.computerworlduk.com/technology/networking/networking/news/index.cfm?newsid=5797.

[3] Hp expands high-performance computing offering with infiniband solutions from cisco. http://www.hp.com/hpinfo/newsroom/press/2007/070524xa.html.

[4] The london stock exchange uses corvil technology to improve on latency and time to market. http://www.corvil.com/products/casestudies/case_study_london_stock_exchange/.

[5] Netfpga. http://www.netfpga.org.

[6] Next-generation routers: A comprehensive product analysis. http://www.heavyreading.com/details.asp?sku_id=662&skuitem_itemid=673&promo_code=&aff_code=&next_url=%2Fdefault.asp%3F.

[7] Openflow. http://www.openflow.org.

[8] YAF: Yet Another Flowmeter. http://tools.netsa.cert.org/yaf/.

[9] ARISTA NETWORKS, INC. 7100 series datasheet. http://www.aristanetworks.com, 2008.

[10] CHOI, B., MOON, S., ZHANG, Z.-L., PAPAGIANNAKI, K., AND DIOT, C. Analysis of point-to-point packet delay in an operational network. In *IEEE INFOCOM* (2004).

[11] COPAS, J. Regression, prediction and shrinkage. *Journal of the Royal Statistical Society. Series B (Methodological) 45*, 2 (1983), 311–354.

[12] DUFFIELD, N. Simple network performance tomography. In *ACM/USENIX IMC* (2003).

[13] DUFFIELD, N., GERBER, A., AND GROSSGLAUSER, M. Trajectory engine: A backend for trajectory sampling. In *IEEE NOMS* (2002).

[14] DUFFIELD, N. G., AND GROSSGLAUSER, M. Trajectory sampling for direct traffic observation. In *IEEE/ACM Transactions on Networking* (2000).

[15] EIDSON, J., AND LEE, K. IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In *Sensors for Industry Conference, 2002. 2nd ISA/IEEE* (2002).

[16] ESTAN, C., KEYS, K., MOORE, D., AND VARGHESE, G. Building a Better NetFlow. In *ACM SIGCOMM* (2004).

[17] ESTAN, C., AND VARGHESE, G. New directions in traffic measurement and accounting. In *ACM SIGCOMM* (2002).

[18] FLOYD, S., AND JACOBSON, V. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking* (1993).

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

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

[21] IYER, S., KOMPELLA, R. R., AND MCKEOWN, N. Designing Buffers for Router Line Cards. *IEEE/ACM Transactions on Networking (ToN) 16*, 3 (2008).

[22] KOMPELLA, R. R., AND ESTAN, C. The power of slicing in internet flow measurement. In *ACM/USENIX IMC* (2005).

[23] KOMPELLA, R. R., LEVCHENKO, K., SNOEREN, A. C., AND VARGHESE, G. Every MicroSecond Counts: Tracking Fine-grain Latencies Using Lossy Difference Aggregator. In *ACM SIGCOMM* (2009).

[24] KOMPELLA, R. R., SNOEREN, A. C., AND VARGHESE, G. mPlane: An architecture for scalable fault localization. In *ACM ReARCH* (2009).

[25] LEE, M., DUFFIELD, N., AND KOMPELLA, R. R. Two Samples are Enough: Opportunistic Flow-level latency estimation using Netflow. In *IEEE Infocom* (2010).

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

[27] 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 JSAC 21*, 6 (2003).

[28] PAXSON, V. End-to-end internet packet dynamics. In *ACM SIGCOMM* (1997).

[29] PAXSON, V., AND FLOYD, S. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking* (1995).

[30] RAMABHADRAN, S., AND VARGHESE, G. Efficient implementation of a statistics counter architecture. In *ACM SIGMETRICS* (2003).

[31] SHANNON, C., ABEN, E., KC CLAFFY, AND ANDERSEN, D. E. CAIDA Anonymized 2008 Internet Traces Dataset (collection).

[32] SOMMERS, J., AND BARFORD, P. Self-Configuring Network Traffic Generation. In *ACM/USENIX IMC* (2004).

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

[34] SOMMERS, J., BARFORD, P., DUFFIELD, N., AND RON, A. Accurate and efficient SLA compliance monitoring. In *ACM SIGCOMM* (2007).

[35] VASUDEVAN, V., PHANISHAYEE, A., SHAH, H., KREVAT, E., ANDERSEN, D. G., GANGER, G. R., GIBSON, G. A., AND MUELLER, B. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *ACM SIGCOMM* (2009).

[36] WOVEN SYSTEMS, INC. EFX switch series overview. http://www.wovensystems.com.

[37] YAN, N. M., CHEN, Y., BINDEL, D., SONG, H., AND KATZ, R. H. An Algebraic Approach to Practical and Scalable Overlay. In *ACM SIGCOMM* (2004).

[38] YUAN, L., CHUAH, C.-N., AND MOHAPATRA, P. ProgME: towards programmable network measurement. In *ACM SIGCOMM* (2007).

[39] ZHAO, Y., CHEN, Y., AND BINDEL, D. Towards unbiased end-to-end network diagnosis. In *ACM SIGCOMM* (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 (PAM 2001)* (2001).