THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

High-Fidelity Per-Flow Delay Measurements with Reference Latency Interpolation

Citation for published version:
Lee, M, Duffield, N & Kompella, R 2013, 'High-Fidelity Per-Flow Delay Measurements with Reference Latency Interpolation', *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1567 - 1580 . https://doi.org/10.1109/TNET.2012.2227793

Digital Object Identifier (DOI):
10.1109/TNET.2012.2227793

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
IEEE/ACM Transactions on Networking

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 13. Nov. 2024

High-Fidelity Per-Flow Delay Measurements with Reference Latency Interpolation

Myungjin Lee, Nick Duffield, Ramana Rao Kompella

***Abstract*—New applications such as soft real-time data center applications, algorithmic trading and high-performance computing require extremely low latency (in microseconds) from networks. Network operators today lack sufficient fine-grain measurement tools to detect, localize and repair 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 while having small overhead, RLI achieves a median relative error of 12% and one to two orders of magnitude higher accuracy than previous per-flow measurement solutions. We also observe RLI achieves as high accuracy as LDA in aggregate latency estimation and RLI outperforms LDA in standard deviation estimation.**

***Index Terms*—Latency, measurements, per-flow, router, switch**

I. 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 [1], storage applications (with industry moving toward Fiber Channel over Ethernet (FCoE) [2]) and, algorithmic trading applications [3] (together constituting multi-billion dollar markets) all require low end-to-end latencies in the order of a 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 [3].

Portions of this manuscript appeared in ACM SIGCOMM 2010. This work was supported in part by the National Science Foundation through award CNS-0831647, and Cisco Systems.

M. Lee is with the School of Informatics, University of Edinburgh, Edinburgh EH8 9AB, UK (e-mail: myungjin.lee@ed.ac.uk).

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

N. Duffield is with AT&T Labs–Research, 180 Park Ave, Building 103, Florham Park, NJ 07932 (e-mail: duffield@research.att.com).

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) in these networks. Once the problem is localized, they can potentially, with fault localization techniques, isolate the particular offending flow or a set of such flows that are 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 [4], [5] provide latency guarantees within 10s of microseconds in a network not having a fan-in traffic pattern—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 with which routers come equipped 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 [6], [7] to infer link and hop properties. Unfortunately, for the granularity of measurements required, active probes need to be injected at an extremely high probe rate making them not suitable for these low-latency networks. Operators in these networks therefore resort to specialized measurement appliances that can be quite expensive; ubiquitous deployment is therefore costly.

Recognizing these challenges, researchers have recently proposed a new high speed router-level data structure called LDA [8] 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 (*e.g.*, a few minutes). 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 [9]). For instance, many soft real-time applications in data centers (*e.g.*, web search, retail, and advertising) rely on Partition/Aggregate workflow and require to meet target deadlines ($\sim$10ms) for each server involved in the workflow [10]. Because the processing time at servers may take up a large portion of the deadline budgets, microbursts can lead to violations against these deadlines. These failures in turn aggravate the quality of the produced results because expired tasks are often cancelled. Diagnosing this kind of problem requires to measure network delays that took for task assignments and result collections. However, obtaining aggregate statistics only (*i.e.*, average latency across several million packets) may be of little use for the diagnosis because the statistics may look normal.

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 [11]. While specific solutions may exist for known problems [11], 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 financial institutions may obtain specific SLAs from service providers (such as guaranteed latency of less than 100 μs) [12]. 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 insights: (1) We observe a significant amount of 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:

• *A measurement study of latency diversity and the temporal localization of delay*. Using real router traces and simulations, we conduct a measurement study (§II) that reveals our main 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 flows within small localized windows constituting a measurement period is similar.

• *An architecture for high-fidelity per-flow latency measurements*. Based on the findings in our measurement study, we propose an architecture (§III) 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 §IV) using a combination of real traces as well as simulations. In our evaluation, the sensitivity on estimation accuracy by different configurations is first studied in §V-A. Then, we observe that our RLI architecture achieves a median relative error of 10-12% (§V-B), and up to two orders of magnitude lower relative error than existing state-of-the-art schemes under specific configurations (§V-C). The comparison between RLI and LDA also shows that they obtain comparatively similar performance (§V-C).

II. Delay Diversity and Locality

Our focus in this paper is to devise a scalable architecture for per-flow latency measurements within a router. These per-flow measurements will enable network operators to *localize* the root cause of any end-to-end latency spikes that customers may complain about. Before we set out to devise such an architecture, it is important to ascertain that one aggregate latency measure (for which efficient solutions such as LDA [8] have already been proposed) is not sufficient. In this section, we show that there exists significant diversity of latency 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 latency diversity within sufficiently short time intervals; we discuss the ramifications of this observation for the design of a scalable architecture. A further conclusion is that common statistics of delays encountered by a stream of active 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. Thus, active probes alone cannot be used to estimate flow-level latencies.

A. Data sets

Given no public traces with synchronized packet timestamps across router ingress and egress interfaces, we resort to two traces: First, we used traces of the passage of synthetic traffic across a real router collected by the authors of [13]. Details about workload and network environment that are used to collect this data set can be found in [13]. Even though the traffic sources are synthetic, they are subject to *real* router forwarding paths, queueing and other behavior, and thus are

Name	#keys	#packets	pkts/key	mean delay (ms)	R
Link: OC-192 (10 Gbps), Duration: 600s, Year: 2008					
CHIC	8.25M	131M	15.9	0.286	9.96e+3
SANJ	10.3M	214M	20.8	0.386	4.54e+4
Link: OC-3 (155 Mbps), Duration: 305s, Year: 2006					
WEB468	140k	2.61M	18.6	0.552	39.0
WEB700	208k	3.98M	19.1	3.70	271

TABLE I
TRACE CHARACTERISTICS: DURATION, NUMBER OF 5-TUPLE KEYS, AVERAGE NUMBER OF PACKETS PER KEY, AVERAGE PACKET DELAY, AND RANGE FACTOR R OF PER-KEY AVERAGE DELAY.

quite realistic in terms of latency. The data set referred to as WISC consists of two traces (WEB468 and WEB700) with different utilization levels, summarized in Table I. Second, we used backbone header traces published by CAIDA [14] 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 also summarized in Table I. 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 traces denoted as CHIC and SANJ represent those collected at Chicago, IL and San Jose, CA respectively. We believe WISC traces and the backbone traces complement each other for us to conduct our study as close to reality as possible.

These four data sets are used to demonstrate the existence of latency diversity and temporal locality of delays experimentally. Later, we resort to the same data sets for the evaluation of our architecture in §V. 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 §IV-A) and real timestamps of packets arrival on an OC-192 interface. Thus we expect that they will provide a realistic representation of the queueing dynamics whose properties underpin our method. By contrast, the fact that WEB468 and WEB700 are obtained through subjection to real router forwarding paths enables us to capture any effects specific to complexities of actual 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 latency diversity.

B. Latency Diversity over Keys

Many studies have found flow arrivals to be bursty (flows do not commence as a Poisson process) and flow durations are heavy-tailed (as opposed to exponentially distributed) [15], [16], [17]. Under such conditions, congestion also tends to be bursty, being concentrated in rarer and longer bursts that would be the case for Poisson traffic. Consequently, the latency 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. Furthermore, 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.

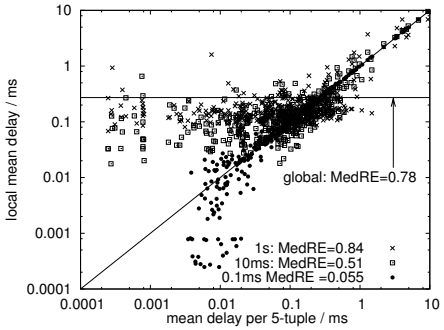
To study delay diversity, we classified each packet according to the standard 5-tuple key comprising source and destination IP addresses and TCP/UDP ports, and IP protocol. Being the finest key definition for our data, this represents the most challenging case for our approach. From Table I we see that WEB700 entails a 50% higher packet rate than WEB468 and the mean delay for WEB700 is about an order of magnitude higher than that for WEB468. SANJ trace comprises a load about 63% higher than CHIC; mean delay for SANJ is about 35% higher than that for CHIC.

We characterize packet delay at the flow level using the per key average packet delay, a simple statistic that is sensitive both to the center and the extremes of the delay values, both of which may influence performance. We characterize the delay diversity over the set of flows by the range factor R (also tabulated in Table I), defined as the ratio of 99th and 1st quantiles in a per-key mean delay distribution. R captures nearly the full extent of the range, while excluding a small number of outliers. For SANJ and CHIC, the range spanned 3 to 4 orders of magnitude, while for WEB468 and WEB700 the range was 1 to 2 orders of magnitude. Unsurprisingly, R was larger for the higher load trace of each pair. The difference in range factors between WISC and OC-192 (CHIC and SANJ) traces is because the range of packet latencies in OC-192 trace is almost three orders of magnitude larger than that of packet latencies in WISC trace. Therefore, 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.

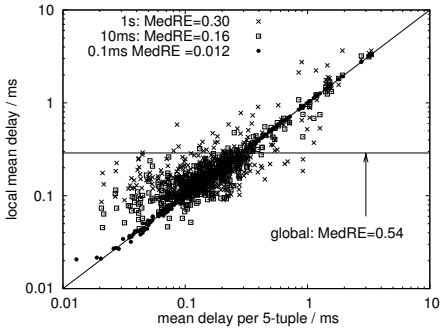
C. Temporal localization of queueing delays

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 [18]. We now demonstrate this empirically, by localizing time, and determining how closely the mean queueing delay experienced by packets of a given flow over a small time window (*e.g.*, up to a few milliseconds) 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 §II-E.

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

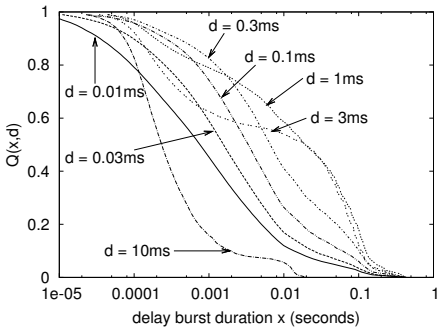


(a) #packets ≤ 10

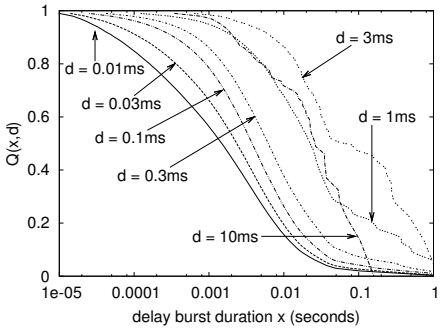


(b) #packets > 10

Fig. 1. Scatter plot of local vs actual mean delay per 5-tuple key with CHIC trace; localization intervals 0.1ms, 10ms, and 1s. Global average delay also shown as horizontal line.



(a) SANJ



(b) CHIC

Fig. 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 $d = (0.01, 0.03, 0.1, 0.3, 1, 3, 10)$ms.

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 $\tilde{d}_i$ weighted by the numbers of packets $n_{i,k}$ for key k in each intervals, should yield a fairly accurate approximation of D_k, 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 key in CHIC, for localization windows of 0.1ms, 10ms and 1s, broken out according to the number of packets per key. For clarity, we show only about 400 randomly selected points for each window localization value. Observe closer agreement for smaller windows, while for large windows the scatter appears to revert to a more horizontal regression, reflecting averaging over longer windows. 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 median relative error (MedRE) 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, so that the accuracy of the localized mean delay is not simply an artifact of comparing a key's packet with itself. Further, Figure 1(a) shows that a small localization window is more beneficial for latency estimation of small flows, while only relative small proportion of small average delays were poorly estimated. Accuracy was found to closer for SANJ than CHIC, presumably a consequence of its higher offered load, as evidenced by the longer mean delays in Table I. Plots for SANJ are omitted for brevity. We now relate these differences specifically to the burst properties of delay episodes.

D. 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 in 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 leads 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. 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).

E. 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 §III 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. Note that our assumption here is that one is interested in only computing the per-flow mean delays; for a deeper understanding of the fluctuations of packet delays, timestamping each packet may still be required. 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 a 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 flow's 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 [18] 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 I, 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 which tend to be more stringent in the latency bounds than a general WAN. For example, a past study [19] 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.

III. 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 [20] 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.

A. 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 average size packets of 125 bytes) could use up to 3.2% 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 [8]. 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, 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 (§II) that packets that belong to different flows experience similar delay when they are closely spaced within each other.

B. 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

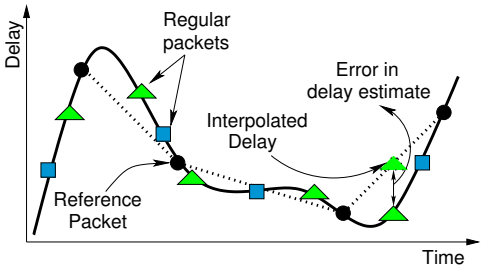


Fig. 3. Key idea in our architecture is to estimate packet delays by interpolating the reference packet latencies.

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.

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 [21] 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.*, $n = 1\,000$). 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 every τ time period (*e.g.*, $\tau = 1$ms). 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

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: $u_{instant} \leftarrow c_b/d_{rp}/l_c$, $c_b \leftarrow 0$
11: $u_{est} \leftarrow u_{instant} \cdot \alpha + u_{est} \cdot (1-\alpha)$
12: $u_{eff} \leftarrow u_{est}$, where $u_{min} \leq u_{est} \leq u_{max}$
13: $r_{eff} \leftarrow \sqrt{1 - (\frac{u_{eff}-u_{min}}{u_{max}-u_{min}})^2}(r_{max} - r_{min}) + r_{min}$
14: **return** r_{eff}
15: **end procedure**

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. An issue with this scheme is that it becomes infeasible to estimate variations in utilization across all links when RLI is deployed across routers. Thus, in this paper, we mainly focus on latency measurements between interfaces within a router.

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 a small amount of state. Specifically, our adaptive scheme consists of two steps: updating link utilization and calculating effective reference packet rate r_{eff}. Algorithm 1 presents the pseudocode for the scheme. The algorithm is triggered immediately after a reference packet is injected with the previously calculated 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, 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 13) 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.

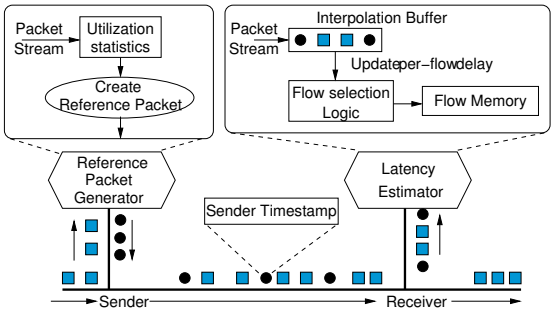


Fig. 4. Overview of our architecture.

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 maintains 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 §VI.)

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 is 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.

C. Packet delay estimators

We formally describe our packet delay estimators in this section. The first estimator called RLI estimator[1] uses 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 (*i.e.*, 8-byte timestamp and 40-byte header) 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.

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

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

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

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

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

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

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 sample. 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,

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

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 [22]. Unfortunately, Shrinkage Estimation provided only a very small improvement in estimation accuracy; we therefore do not consider this further in this paper.

IV. Evaluation methodology

This section outlines the methodology we used to evaluate our architecture. Experimental results are presented in §V.

A. Simulator

We evaluate our architecture using a simulator we create by extending an open-source NetFlow platform called YAF [23] 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. While we simulate RED queue management strategy as well as DropTail queue, we observe little difference in evaluation results between them. In this paper, we only present results using RED in interest of space.

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. The drain rate is defined as bytes per second. By fixing the drain rate less than 1.25GB/s, we can automatically control both the delay as well as the loss distribution. Following the guidelines in [24], we chose a queue size of 10 000 [2], $min_{th} = 4\,000$ 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 our simulation using CHIC and SANJ traces is open-loop, *i.e.*, we do not see TCP backoff effects even when packets are dropped. By contrast, both WEB468 and WEB700 traces are generated by configuring a real router with RED, and as such, they 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

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

[2] Assuming 10 Gbps link and 1ms RTT in a data center network, the rule-of-thumb (*i.e.*, bandwidth-delay product) suggests a router buffer of 10 Mbits. Assuming average packet size be 125 bytes, we choose 10 000.

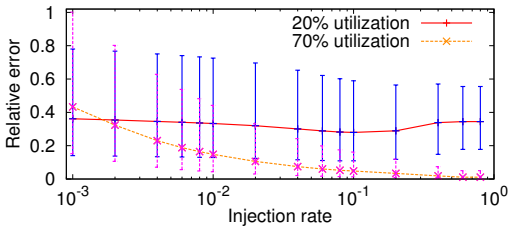


Fig. 5. Accuracy changes of delay mean estimates over injection rate variation. Each curve represents median relative error and error bars show 25th and 75th percentiles.

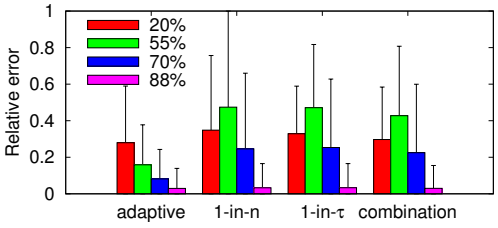


Fig. 6. Comparison of reference packet injection methods. In the figure, combination means the method to combine 1-in-n and 1-in-τ methods. Bar in histogram denotes median relative error and whisker indicates 75th percentile. Percentage in legend indicates link utilization.

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 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.

B. Other solutions for comparison

Trajectory sampling. First, we consider trajectory sampling proposed by Duffield *et al.* to sample packet trajectories [25]. 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 §III).

Multiflow estimator. Next, we consider a new estimator called Multiflow estimator (MFE) proposed by Lee *et al.* in [18]. 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 main spirit is grounded in a similar observation as ours in §II.

LDA. This solution is designed to obtain aggregate latency statistics over a measurement period (say 1 second). We compare RLI with LDA to evaluate the performance of RLI for obtaining the statistics and the tradeoff between them.

V. Results

We divide our results into four major parts: First, we evaluate reference packet generator in terms of injection rate and mechanism. Second, we evaluate the accuracy of our RLI estimator, both mean and standard deviation estimates, for different traces and different utilizations. Third, we compare our architecture with other solutions such as the trajectory sampling, MFE, and LDA described in §IV-B. Finally, we evaluate the overhead involved in our architecture.

As a primary metric to evaluate the accuracy of RLI, we focus on the relative error (defined as $|true - estimated|/true$) of mean and standard deviation estimations of each flow with the ground truth. In addition to relative error, sometimes absolute error serves as a secondary metric to provide deeper understanding about our results (*e.g.*, high relative error under low utilization discussed in §V-B).

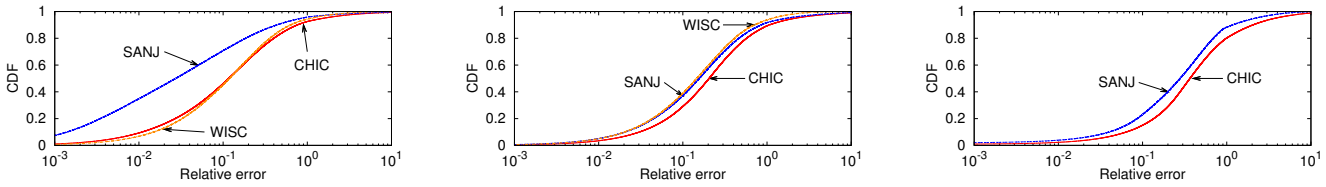
Although our experimental settings and scenarios cover essential performance issues of RLI, we note that a few additional experiments and analyses will be also useful. One such experiment is understanding absolute errors and impact of parameterization on accuracy with real workloads captured from various vantage points (*e.g.*, ToR, aggregate, and core switches) in a data center.

A. Impact of reference packet injection rate and mechanisms

We investigate two different aspects of reference packet generator: injection rate and mechanism. By changing values of these two knobs, we observe with SANJ trace how the accuracies of per-flow latency estimates are impacted.

Impact of injection rate. We statically configure injection rate to study the impact of injection rate. While there are four injection mechanisms, we use 1-in-n mechanism because of its simplicity for controlling injection rate. We vary n accordingly to control reference packet injection rate from 0.001 to 0.8. We test out these injection rates under several different utilization conditions. In general, as we increased injection rate, the accuracy of per-flow latency estimates was improved. We only show two interesting graphs that illustrate 20% and 70% link utilization cases. The curve for 20% utilization in Figure 5 shows very slow decrease in relative error that remains high (around 28-36% median relative error) regardless of injection rate. At higher injection rate than 20%, we observe that the variance of errors is narrower than any other injection rate cases. Low accuracy under low link utilization stems from the fact that average per-flow latency is just a few microseconds, serialization time is a dominant factor of packet delay, and queueing delay is negligible. The serialization time is more dependent on the packet size, which in turn causes more jitter in interpolation process. Thus, our estimators yield low accuracy because even high injection rate is hard to capture serialization time accurately.

Compared to 20% link utilization, the curve of 70% link utilization shows a quite different pattern where as injection rate increases, errors rapidly decrease. At the begining, median relative error is about 43%, but at 10% injection rate, median relative error is only about 5% (8x reduction).



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

Fig. 7. CDF of mean per-flow delay estimates using RLI estimator for different utilizations and different traces.

Impact of injection mechanism. We evaluate which injection method is more effective to obtain high accuracy across different link utilization situations out of the four methods discussed in §III-B1. One may set injection rate of static schemes as large as 10% for higher accuracy. However, in our experiment, this high injection rate caused 22% increase in packet loss rate (0.67% loss rate with no injection and 0.82% with 1-in-10 injection method) at 90% link utilization. To avoid such a high interference, we set $n = 300$ for 1-in-n scheme and τ is set to $780\mu s$ for 1-in-τ. These parameters were chosen to meet the injection rate (*i.e.*, 0.35%) that adaptive scheme achieves at 90% link utilization. The combination scheme of 1-in-n and 1-in-τ uses the same values of n and τ, which yields 0.51% injection rate. For the adaptive scheme, we use parameters described in §III-B1. The injection rates by the parameters are 10% (at 20% utilization), 9% (at 55%), 4.8% (at 70%) and 0.6% (at 88%).

Figure 6 shows the accuracy of each injection mechanism. We first see that the adaptive scheme achieves the best accuracy among others. In general, as link utilization increases, the difference in accuracy among all the schemes decreases. On the contrary, when link utilization is 55%, the other schemes have at least 40% median relative errors, but the adaptive scheme achieves 50% less median relative error than the others. We expected the combination scheme would work as good as the adaptive scheme, but the scheme just achieves slightly better accuracy than 1-in-n and 1-in-τ schemes. The highest error by the static schemes at 55% utilization—where delay variability is high—is another evidence of inflexibility of those schemes. The adaptive scheme injects more number of reference packets than the static schemes when a link is less loaded, and hence imposes a higher overhead than the others. The static schemes, however, are inflexible to respond to the variation of link utilizations, thus leading to either low accuracy or high interference with regular traffic. Therefore, in the rest of evaluations, we only use the adaptive scheme. Note that §V-D discusses the overhead of RLI in greater detail.

B. Accuracy of RLI

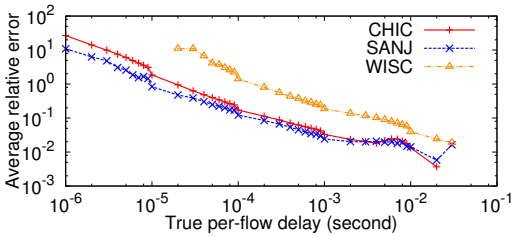
Accuracy of mean latency. We plot the cumulative distribution function (CDF) of the relative error of mean delay estimates for all the flows in Figure 7 for different utilizations and traces. In our evaluation, we consider the WEB468 as a moderate utilization scenario with about 55% link utilization, 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 7(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 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. Recall that this difference stems from the fact that, under low utilization, serialization time takes more portion in latency and serialization time causes more jitter in interpolation process.

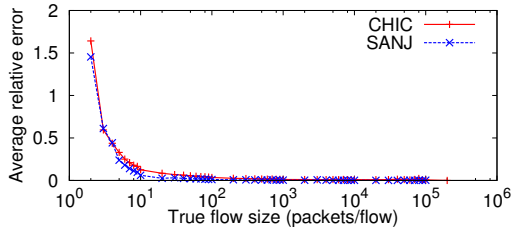
We envision that our architecture is more suitable for isolating the router where a flow experienced bad end-to-end latency; we therefore care about flows for which the delays are significantly higher than the rest. In other words, for small delays (*e.g.*, $10\mu s$), a relative error of 100% is not nearly as significant as compared to flows which experience higher delays (*e.g.*, $100\mu s$). 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 8(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 $90\mu s$ and $10\mu s$ 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 $100\mu s$ all the way until about 30ms.

We can observe from Figure 8 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 $100\mu s$ (in the SANJ trace). For the CHIC trace, we found that 75% of flows having about $100\mu s$ latency have less than 18% relative error, 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



(a) Binned by delay



(b) Binned by flow size

Fig. 8. Average relative error of per-flow delay mean estimates binned by true flow delays and sizes.

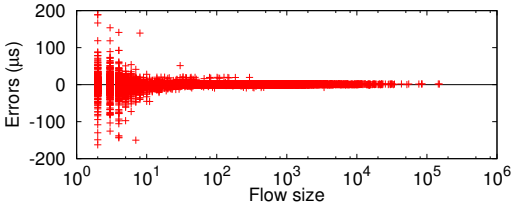
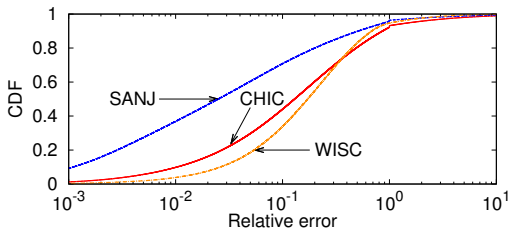
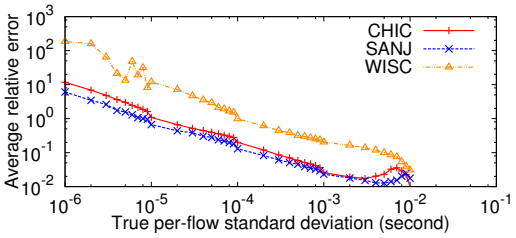


Fig. 9. Unbiased nature of RLI using SANJ trace.



(a) CDF



(b) Binned by standard deviation

Fig. 10. Average relative error of per-flow delay standard deviation estimates binned by true deviations.

bounded; thus, our solution can be quite effective in measuring flow-specific delay spikes of the order of a few $100\mu s$ very efficiently—exactly the level of SLA specifications that Cisco provides in its trading floor architecture [12].

Our real router trace, WISC, shows similar trends with CHIC and SANJ in that as true delay increases, 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, $100\mu s$ 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 8(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.

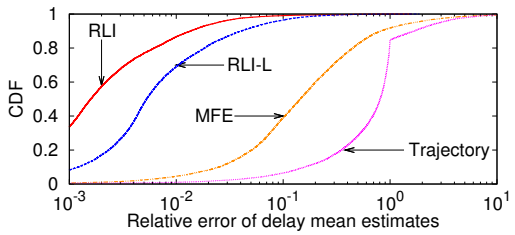
Unbiased nature of RLI. Per-flow latencies are computed from approximated packet latencies, which may potentially cause biased estimates. Hence, we empirically demonstrate the unbiasedness of estimates by RLI in Figure 9; we can clearly observe positive as well as negative errors in y-axis. We further notice that as the flow size increases, the error ranges become narrow, thus reassuring the efficacy of RLI for large flows.

Accuracy of delay standard deviation. While good accuracy in average latency estimates is nice, it is important to be able to estimate 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 10(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 10(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 smaller.

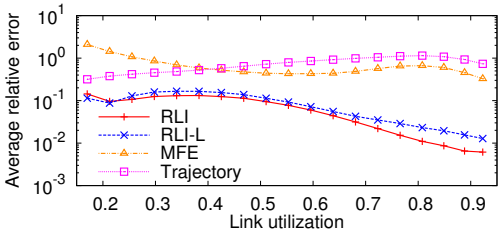
C. Comparison with other solutions

We largely conduct two different sets of experiments to compare with other solutions. The first set is to evaluate per-flow latency measurements and the second set is to understand the performance of RLI for aggregate latency statistics.

Per-flow latency. We compare our architecture with previously proposed solutions discussed in §IV-B, trajectory sampling and MFE, and also study the advantage of using RLI estimator



(a) CDF of relative error



(b) Average relative error by utilization

Fig. 11. Comparison with other solutions on SANJ trace with a packet sampling rate of 0.1%.

compared to RLI-L estimator. In these experiments, we introduce a sampling rate of 0.1% in order to keep the trajectory sampling overhead 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 [26]) 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 11(a) for the high 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 a 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 11(b). We again observed that RLI

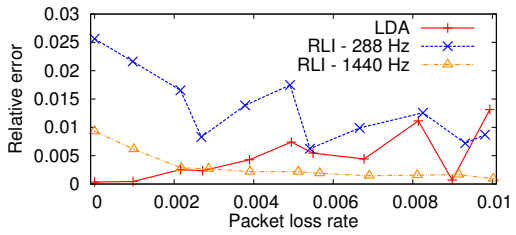


Fig. 12. Aggregate latency estimation accuracy of RLI and LDA.

outperforms the rest significantly; the gap between RLI and the rest is 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.

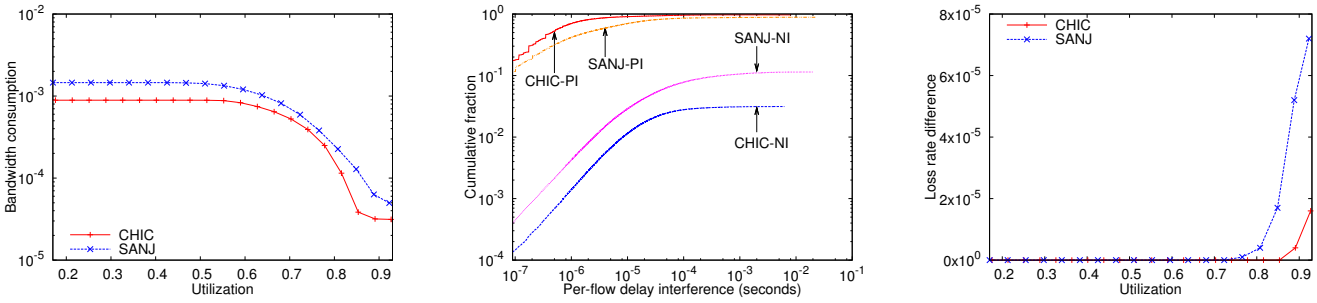
Aggregate latency. While RLI is designed for per-flow estimates, one could use the packet latency estimates to compute aggregate statistics as well, for which LDA has been proposed. We use the two-bank LDA suggested in [8] with one tuned for 0.5% loss rate and the other for 10%. For RLI, two settings are used: (1) 288 and (2) 1 440 reference packets per interval. While the former setting consumes the same communication bandwidth that LDA requires, the latter roughly translates to 0.34%, the lowest injection rate used in our experiments. For the experiment, we use a measurement interval of 1s in which about 0.4 million packets arrived (for the SANJ trace).

Figure 12 shows the aggregate latency estimation accuracy of three estimators (LDA, RLI-288 Hz, and RLI-1440 Hz) depending on different packet loss rate. We find two observations from the figure. Under the same bandwidth constraint (LDA and RLI-288 Hz), LDA performs better than RLI while RLI still achieving less than about 2.5% relative error. In comparison of LDA and RLI-1440 Hz, however, there is a unique trade-off. When packet loss rate is less than 0.4%, LDA yields better accuracy than RLI. On the contrary, as packet loss rate becomes larger than 0.4%, RLI stabilizes relative error by achieving less than 0.3% relative error, but the accuracy of LDA varies significantly ranging from 0.1% to 1.3%. The stable accuracy of RLI is based on the fact that it estimates delays of all the incoming packets at a receiver, and latencies of individual packets becomes more similar as queue gets overloaded (as a result, high packet losses occur). On the other hand, the pattern of LDA is related to tuned packet loss rates in two banks and hence varies within increasing loss rates.

For variance, unlike average delay, RLI experiences about three orders of magnitude lower relative error than LDA. For instance, while the relative error of RLI is about 0.005% all the time, the relative error of LDA only varies between 1% and 27%. In interest of space, we omit showing the exact graph.

D. Overhead of RLI

We quantify the direct and obvious overhead associated with the reference packet traffic, and indirect effects of the reference packet traffic on actual per-flow latencies and losses. In Figure 13(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



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

Fig. 13. Quantifying overhead in RLI and the interference of reference packet packets on regular flows.

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 by a flow with and without our architecture. In Figure 13(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), but, we also found about 10% flows for which the average delay went down (curves x-NI, NI is negative interference). Upon investigation, we found a significant variation in the number of packets that were dropped from flows; 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 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, RLI introduces very little increase in the loss rate as can be seen in Figure 13(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.

VI. Implementation

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. 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 [27] and packet buffers [28], 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 [29]).

VII. 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. Tomography techniques (*e.g.*, [6], [7], [30]) have been proposed in the past to infer hop and link characteristics from end-to-end measurements (conducted using tools such as [31]) 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.*, [29], [26], [32], [33]) 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. Researchers have in the past conducted measurement studies to understand single-hop delays [34], [35] and delays across PoPs in [19]. They do not propose any architecture for measuring per-flow delays however.

The three most relevant works are trajectory sampling [25], Multiflow estimator [18] and LDA [8]. Because of their relevance, we have already described them in great detail and compared them with our approach.

VIII. Conclusion

Many new applications such as algorithmic trading and data center applications demand low end-to-end latency in the order of microseconds. 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 two 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% (§V-B), and one to two orders of magnitude lower relative error compared to previous solutions for per-flow latency measurements (§V-C). For aggregate latency measurements, we observe that RLI obtains a comparable accuracy of LDA (§V-C). 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.

REFERENCES

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

[2] INCITS, "Fibre channel backbone-5 (FC-BB-5)," Oct. 2008, ver. 1.03.

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

[4] Arista Networks, Inc., "7100 series datasheet," http://www.aristanetworks.com, 2008.

[5] Woven Systems, Inc., "EFX switch series overview," http://www.wovensystems.com.

[6] Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An Algebraic Approach to Practical and Scalable Overlay Network Monitoring," in *ACM SIGCOMM*, 2004.

[7] N. Duffield, "Simple network performance tomography," in *ACM/USENIX IMC*, 2003.

[8] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese, "Every MicroSecond Counts: Tracking Fine-grain Latencies Using Lossy Difference Aggregator," in *ACM SIGCOMM*, 2009.

[9] "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.

[10] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[11] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *ACM SIGCOMM*, 2009.

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

[13] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Improving accuracy in end-to-end packet loss measurement," in *ACM SIGCOMM*, 2005.

[14] C. Shannon, E. Aben, kc claffy, and D. E. Andersen, "CAIDA Anonymized 2008 Internet Traces Dataset (collection)."

[15] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking*, 1995.

[16] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation*, 1st ed. John Wiley & Sons, Inc., 2000.

[17] S. Uhlig, "Non-stationarity and high-order scaling in tcp flow arrivals: a methodological analysis," *ACM Computer Communication Review*, vol. 34, pp. 9–24, 2004.

[18] M. Lee, N. Duffield, and R. R. Kompella, "Two Samples are Enough: Opportunistic Flow-level latency estimation using Netflow," in *IEEE Infocom*, 2010.

[19] B. Choi, S. Moon, Z.-L. Zhang, K. Papagiannaki, and C. Diot, "Analysis of point-to-point packet delay in an operational network," in *IEEE INFOCOM*, 2004.

[20] J. Eidson and K. Lee, "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.

[21] V. Paxson, "End-to-end internet packet dynamics," in *ACM SIGCOMM*, 1997.

[22] J. Copas, "Regression, prediction and shrinkage," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 45, no. 2, pp. 311–354, 1983.

[23] "YAF: Yet Another Flowmeter," http://tools.netsa.cert.org/yaf/.

[24] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, 1993.

[25] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," in *IEEE/ACM Transactions on Networking*, 2000.

[26] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *ACM SIGCOMM*, 2002.

[27] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," in *ACM SIGMETRICS*, 2003.

[28] S. Iyer, R. R. Kompella, and N. McKeown, "Designing Buffers for Router Line Cards," *IEEE/ACM Transactions on Networking (ToN)*, vol. 16, no. 3, 2008.

[29] L. Yuan, C.-N. Chuah, and P. Mohapatra, "ProgME: towards programmable network measurement," in *ACM SIGCOMM*, 2007.

[30] Y. Zhao, Y. Chen, and D. Bindel, "Towards unbiased end-to-end network diagnosis," in *ACM SIGCOMM*, 2006.

[31] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Accurate and efficient SLA compliance monitoring," in *ACM SIGCOMM*, 2007.

[32] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a Better NetFlow," in *ACM SIGCOMM*, 2004.

[33] R. R. Kompella and C. Estan, "The power of slicing in internet flow measurement," in *ACM/USENIX IMC*, 2005.

[34] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot, "Analaysis of measured single-hop delay from an operational backbone network," *IEEE JSAC*, vol. 21, no. 6, 2003.

[35] N. Hohn, D. Veitch, K. Papagiannaki, and C. Diot, "Bridging router performance and queuing theory," in *ACM SIGMETRICS*, 2004.

Myungjin Lee (M'12) is currently an Assistant Professor and Chancellor's Fellow in the School of Informatics at the University of Edinburgh. He received his Ph.D. degree from Purdue University in 2012, M.S. from KAIST in 2002, and B.E. from Kyungpook National University, Korea in 2000. His major research focuses on scalable measurement architectures and algorithms for data center networks, scheduling resources in cloud networks, and networked application monitoring and characterization.

Nick Duffield (M'97, SM'01, F'05) received the Ph.D. degree from the University of London, London, U.K., in 1987. He is a Distinguished Member of Technical Staff and an AT&T Fellow in the Internet and Network Systems Research Center, AT&T Labs-Research, Florham Park, NJ, where he has been since 1995. He previously held post-doctoral and faculty positions in Dublin, Ireland, and Heidelberg, Germany. He is a co-inventor of the Smart Sampling technologies that lie at the heart of AT&Ts scalable Traffic Analysis Service. His current research focuses on measurement and inference of network traffic. Dr. Duffield was Charter Chair of the IETF working group on Packet Sampling. He is an Associate Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING.

Ramana Rao Kompella (M'07, ACM M'07) is currently an Assistant Professor in the Department of Computer Sciences at Purdue University. His main research interests include fault-management in IP networks, scalable algorithms and architectures for high speed switches and routers, and scheduling in wireless networks. He received his Ph.D degree from UCSD in 2007, M.S from Stanford University in 2001, and B.Tech degree from IIT Bombay in 1999.