Enabling Flow-level Latency Measurements across Routers in Data Centers

Parmjeet Singh, Myungjin Lee, Sagar Kumar, Ramana Rao Kompella
Purdue University

Abstract

Detecting and localizing latency-related problems at router and switch levels is an important task to network operators as latency-critical applications in a data center network become popular. This however requires that measurement instances must be deployed at each and every router/switch in the network. In this paper, we study a partial deployment method called Reference Latency Interpolation across Routers (RLIR) to support network operators' requirements such as incremental deployment and small deployment complexity without losing localization granularity and estimation accuracy significantly.

1 Introduction

As many data centers today host latency-critical applications such as web search, retail advertising and recommendation systems [3], network operators are consciously trying to guarantee low end-to-end latency. End-to-end latency observed by clients consists of two main components—between client and the data center, and within the data center. To reduce the latency between the client and data centers, many service providers today heavily rely on infrastructures like Content Distribution Networks (CDN) where their customers' traffic is served at the data center closest to customers' networks as possible to minimize latency. In this paper, we are concerned mainly with the other component, namely, latency within a data center.

In data center networks, tasks assigned to clusters usually need to meet stringent latency requirements. For example, Google's instant search service [1] provides search results as a user types key words. This implies that a search query sent to Google's data centers needs to be processed within a few 100ms assuming about 300ms period between keystrokes. Then, each network components such as routers and switches in the data center network only have tens of μseconds to forward requests to a cluster as majority time is consumed to process the query in the cluster. Thus, high network latency significantly aggravates customer experience on the service which possibly decreases revenue of service providers like Google. Similarly, in financial data center networks, message latencies are quite critical since automated trading platforms respond to stock price fluctuations to buy a stock cheaply and sell it when the price goes up. Failing to guarantee low message latencies can potentially cause millions of dollars in lost arbitrage opportunities.

Operators running data centers need mechanisms to detect and diagnose any latency abnormalities suffered by applications. Most applications involve several layers of software, I/O, and accesses over the network. While end-host latencies are typically monitored locally by the operating system, detecting and diagnosing abnormal latencies through the network is more complicated, since there exists no native support in latency measurements in today's commodity switches.

Two prior works, LDA [9] and RLI [11], provide switch-based mechanisms to obtain high-fidelity measurements at the microsecond granularity. Out of the two, RLI provides measurements on a per-flow measurements while LDA provides only aggregate measurements. Thus, the fact RLI provides more fine-grained measurements than LDA, makes RLI a more suitable architecture for the kind of latency measurement support required in the aforementioned environments. At a high level, RLI basically works by injecting a reference packet that carries a timestamp within the packet periodically between the measurement points, and uses linear interpolation to estimate the latency of packets between the two reference packets. The fundamental premise behind the idea of linear interpolation is delay locality, whereby two packets that traverse a path segment roughly at the same time will potentially experience similar (or correlated) delays. This delay locality assumption forms the key underpinning of RLI.

While [11] has shown that it produces accurate per-flow latency measurements, deployment of RLI is a significant challenge, since it requires *all* switches in the network to implement RLI. In this paper, we consider a *partial deployment* of RLI architecture within a network, that involves implementing RLI only in some routers, or more precisely, implementing RLI *across* routers (RLIR). By only upgrading a few routers to implement RLI, we can considerably reduce the deployment costs, but the disadvantage is that there will be an increase in the localization granularity, i.e., the size of the segment to which we can localize the latency spikes. We believe this is an acceptable tradeoff for already established data centers.

The original requirement in RLI for upgrading all routers stems from the delay locality assumption, because between a (series of) queue(s) within a router, the assumption typically holds true. However, implementing RLI across routers in a data center is challenging since the assumption may not hold true in most data centers that use multipathing across two routers. Specifi-

cally, packets may potentially take different paths across a given pair of routers since routers use equal-cost multipath (ECMP) forwarding. In such cases, delay locality may not hold true since the delay of a reference packet that traverses one path may have no correlation with the delay of a packet that traverses a different path.

Another issue that comes up in deploying RLI architecture across routers is that of cross traffic. A sender in RLI architecture adapts its reference packet injection rate based on an estimated link utilization at the interface. Unfortunately, the sender cannot easily estimate utilization across routers, because it has no idea about the amount of cross traffic at intermediate routers.

We deal with these two issues using the following ideas. First, we deploy RLI instances in every other switch. RLI instances selectively compute per-flow latency measurements by finding regular packets and reference packets that traversed the same path (Section 3). Second, to deal with cross traffic, we essentially assume worst case utilization. We quantify the impact of this assumption through experiments in Section 4. We observe that RLIR architecture achieves median relative error of 4.5% for per-flow latency estimates in the presence of cross traffic yielding 93% link utilization while incorrect link utilization estimation does not cause too much interference with regular traffic in terms of packet loss rate.

2 Background

We first provide an overview of RLI architecture before we discuss our scheme to support per-flow latency measurements across routers. The basic premise behind RLI is that, within bursts of delay, packets belonging to different flows experience similar queueing delays. For a pair of interfaces in a switch, two RLI instances (a sender and a receiver) are installed in each interface along the forwarding path. Time-synchronization between RLI instances is a basic requirement, that can be achieved by GPS-based clock synchronization or IEEE 1588 [8]. An RLI sender regularly injects special packets called reference packets that carry a (hardware) timestamp to an RLI receiver. The RLI receiver then easily obtains *true* delays of these special packets based on the local clock.

The delay samples can then be used to approximate the latency of regular packets. Specifically, consider a train of regular packets arriving at an RLI receiver between two reference packets. (We first see a reference packet followed by regular packets and then the second reference packet.) Given the delays of the two reference packets (computed from the timestamps), and arrival times of the reference and regular packets, RLI uses linear interpolation to estimate per-packet latency. Note here that the assumption is that regular packets cannot carry a timestamp since that would require intrusive changes to router forwarding paths; if they could, there is no need

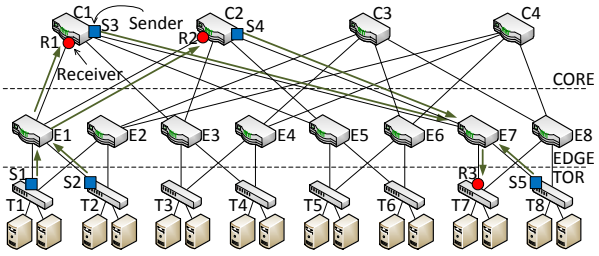


Figure 1: A data center network architecture with partial placement of measurement instances.

for reference packets. The idea that RLI uses, i.e., only a small number of known delays is sufficient to estimate the delays of individual packets, works because the delay locality assumption holds true in general. Obtaining per-flow measurements now is just a matter of aggregating latency estimates across packets that share a given flow key. For further details, see [11].

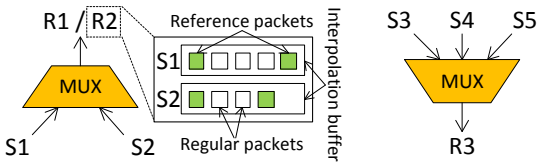
3 RLI across Routers

Consider the problem of detecting and localizing latency anomalies of all flows traversing paths between a pair of interfaces in ToR switches $T1$ and $T7$ (say, an interface hosting an RLI sender, $S1$, and an interface hosting an RLI receiver, $R3$) from Figure 1. The most effective deployment strategy is to install RLI instances at every interfaces of switches/routers that packets can traverse from $T1$ to $T7$ through. Due to its associated difficulty and cost, however, network operators may want to deploy the solution partially. We propose an architecture called RLIR (short for Reference Latency Interpolation across Routers), which requires deploying RLI instances at fewer number of routers, for a small increase in the localization granularity (similar to mPlane [10]). For example, from Figure 1, we can divide the path between $T1$ and $T7$ into segments $T1 - C1$ and $C1 - T7$ which will reduce the number of upgraded routers from 5 to 3.

However, deploying RLIR faces two unique challenges; 1) traffic multiplexing, and 2) cross traffic. While we mainly deal with the problem and solution of traffic multiplexing in this section, we briefly discuss the impact of the presence of cross traffic and evaluate it empirically in Section 4.

3.1 Traffic Multiplexing

Correct operation of RLI (as discussed in Section 2) requires applying linear interpolation for packets that traversed exactly the same path as reference packets. In our context, however, there is a possibility of traffic multiplexing, where other flows that only partially share the path that reference packets traversed may get multiplexed with packets that fully share the path. For instance,



(a) Upstream (b) Downstream

Figure 2: Different traffic multiplexing situations depending on location of an RLI receiver.

in Figure 1, if we are interested in per-flow measurements across $S1$ and $R1$, packets from $T2$ to $C1$ via $E1$ may multiplex with those from $T1$ to $C1$. Depending on the direction of traffic multiplexing, there are two possibilities—upstream and downstream—that are similar in nature but require separate solutions.

Upstream. The issue here is that packets from different senders may end up at the same receiver; thus, many RLI senders need to associate with a given RLI receiver, and the receiver needs a mechanism to distinguish both regular and reference packets to isolate the streams. The simplified version of the upstream multiplexing case in Figure 1 is shown in Figure 2(a). In our example fat-tree topology, $R1$ may have two senders, $S1$ and $S2$, that need to be properly demultiplexed, otherwise per-flow latency estimates at the receivers can be totally wrong.

We address this issue with two ideas. First, each sender sends reference packets to *all* intermediate receivers through which its packets may cross. For example, $S1$ must send reference packets to both $R1$ and $R2$, and so does $S2$. Second, an RLI receiver needs to know the source of the reference packets and regular packets. The RLI receiver can identify reference packets' origin easily via an RLI sender ID (or IP address of the interface which $S1$ sitting on). While regular packets are not as easily identifiable, note that in many cases (such as the fat-tree example), the origin of regular packets can be easily identified by IP address block assigned for hosts in each ToR switch. Thus, upstream RLI receivers need to perform simple IP prefix matching.

Downstream. The issue of downstream multiplexing is much more complicated than that of the upstream case because of two reasons. First, the unique paths taken by regular packets coming from intermediate (*e.g.*, core) switches cannot be determined by IP prefix matching any more. For instance, regular packets from $T1$ via the interface consisting of $S1$ can reach $R3$ via either $C1$ or $C2$. In such a case, IP prefix matching alone is not sufficient. Second, the receivers have to deal with some amount of upstream multiplexing as well. For example, in Figure 2(b), while $S3$ and $S4$ are downstream RLI senders, $S5$ is an upstream sender, a subset of whose packets may still end up at $R3$.

We address the issues of downstream multiplexing depending on the type of RLI sender. For identifying an upstream sender, we can simply use the prefix-matching trick discussed in the upstream case. From the example, regular packets from a ToR switch $T8$ to $R3$ can be identified by IP prefix matching. For identifying the intermediate routers through which packets are routed to the receiver, we can exploit either of two approaches; i) packet marking and ii) reverse ECMP computation. Packet marking is a simple way to address the issue, where the type-of-service (ToS) field in the IP header could be used to mark packets, similar to prior solutions for IP traceback [13]. While this is certainly an easy approach, it requires some native packet marking support from core routers, which may or may not be feasible.

The other approach is to leverage the routing information to isolate the exact path a given packet may have taken from the source router (thereby identifying the intermediate router through which it has passed). However, routers typically use ECMP forwarding where a packet's source and destination IP addresses are typically hashed to identify the next hop. While typically switch vendors do not actually publish the hash functions, we can potentially persuade the switch vendors to reveal them, in which case, we can 'reverse' engineer the intermediate router through which a packet may have originated.

In our example, $R3$ uses the hash functions of edge routers connected to core routers to determine to which core router a particular packet is forwarded from possible edge routers. In order to decide if a regular packet arrives at $R3$ through $S3$, $R3$ could use the hash function of edge routers $E1$, $E3$ and $E5$ for the packet, because these edge routers are connected to core routers $C1$ and $C2$. If the uplink number decided for forwarding the packet is one connected to $C1$, we associate the packet with reference packets from $S3$. This become definitely more cumbersome than the packet marking approach, but requires fewer firmware changes in the routers.

Partial Placement Complexity. We now discuss the complexity of our approach in terms of number of routers to upgrade. Consider k-ary fat-tree topology. We assume that each measurement instance can play a dual role of a sender and a receiver. In RLIR, measurement instances are only installed at interfaces of core routers and upper ones of ToR switches. If we are only interested in a particular pair of two interfaces at different ToR switches like ($S1$, $R3$) pair, we need to install two measurement instances at $k/2$ core routers and an instance at each ToR switch ($S1$ and $R3$). In total, we need $k+2$ instances. For measurements between only two ToR switches, we need $k(k+2)/2$ instances ($k^2/2$ at core routers and k at ToR switches). More generally, in case where we measure per-flow latencies for every pair of ToR switches, $(k/2)^2 k$ instances at all core routers are required and $k/2$

ToR switches need to install $k/2$ measurement instances, totaling $(k/2)^2(k+1)$. On the other hand, for full deployment, there are k-pods, each containing k switches, thus installing two instances for each pair of interfaces in each switch or router requires $O(k^4)$ and the same order of instances are required at the core level. Thus, depending on the deployment granularity (*e.g.*, a pair of two ToR switch interfaces, a pair of two ToR switches, and so forth) that network operators want to achieve, they can control the complexity of deployment.

3.2 Cross Traffic

The existence of cross traffic potentially impacts on the performance of active one-way measurement architecture in two ways. First, it may cause inefficiency to adapt active probe rate based on the offered traffic load at a sender as RLI adapts reference packet injection rate based on an estimated utilization at the sender. Thus, if estimated link utilization is small at the sender, it increases injection rate, but adaptation may fail because it does not know the presence of cross traffic and injects too many reference packets. Second, cross traffic may significantly alter delay statistics compared to ones that one can observe when there is no cross traffic. Hence, it may aggravate the accuracy of per-flow latency measurements. To address this issue, we essentially assume worst case link utilization and inject RLI reference packets statically according to the lowest possible rate required for reasonable accuracy. However, we evaluate both adaptive and static injection schemes for the comparison purpose in Section 4.

4 Evaluation

We evaluate performance of RLIR architecture in the presence of cross traffic across multiple hops. A performance metric is the relative error. Another focus is to study the amount of interference due to high reference packet injection rate caused by the unknown utilization of a bottleneck link. We begin our discussion with simulation environment.

4.1 Simulation Environment

Our simulation environment consists of trace and simulator. A trace is fed into the *in-house* simulator based on an open-source NetFlow software–YAF [2]. The simulator basically lets packets from the trace experience processing and queueing delays across multiple queues (equivalently, multiple routers/switches) and estimates per-flow latencies. We provide a detailed explanation of each component in the following.

Trace. We use two 1 minute traces collected from an OC-192 link [14]; one for regular traffic and the other for cross traffic. We modify IP addresses of cross traffic to distinguish from regular traffic. Regular traffic is one

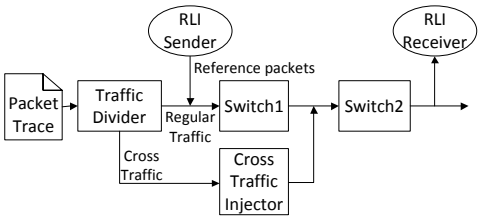


Figure 3: Simulation environment with two hops.

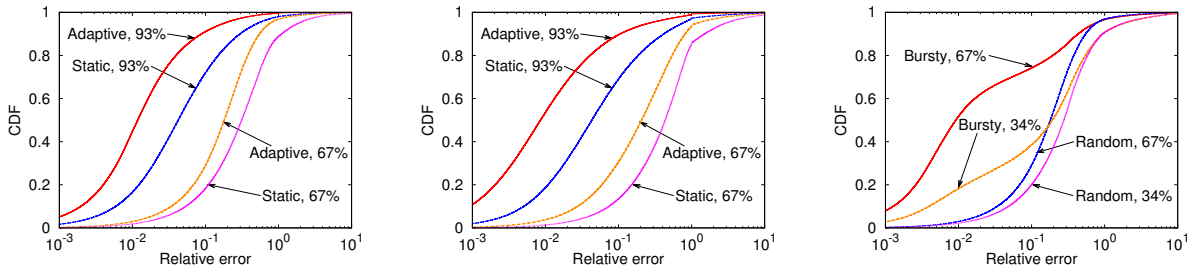
that traverses through a switch where an RLI reference packet generator (RLI sender) is running on. Cross traffic does not traverse through the switch but shares same subpath with the regular traffic to the switch where an RLI latency estimator (RLI receiver) is running. The basic purpose of cross traffic is to adjust link utilization at an intermediate node for controlled experiments.

Some statistics about regular traffic are as follows. The number of packets is about 22.4M and the number of flows is about 1.45M. For cross traffic, there are 70.4M packets. To control link utilization at an intermediate node, we control the number of cross traffic packets with different cross traffic injection models. We discuss these models later.

Simulator. A simple block diagram for our simulator is shown in Figure 3. The simulator reads a packet trace and classifies packets as either regular traffic ones or cross traffic ones based on IP addresses. Regular traffic is directly injected to the queue of the first node and becomes subject to processing and queueing delays, which are governed by queue size and packet processing time. On the other hand, cross traffic arrives at the cross traffic injector module and cross traffic injection rate is controlled by the module. The controlled injection rate eventually determines link utilization at the queue of the second node (the bottleneck node) along with regular traffic from the first switch.

The cross traffic injector provides two types of traffic selection models; uniform and bursty models. Uniform model randomly selects cross traffic with a given probability, which can demonstrate a persistent congestion event as we increase injection rate. Bursty model simulates a situation where cross traffic arrives in a bursty fashion by controlling cross traffic injection duration.

An RLI sender can inject reference packets statically or adaptively. Static injection scheme is a way to inject a reference packet after every n regular packets, which we call 1-and-n scheme. Adaptive scheme dynamically adjusts the injection rate based on the link utilization of a link where the sender is running. The injection rate is controlled by a decreasing function of link utilization. We configured adaptive scheme suggested in [11] by letting injection rate vary between 1-and-10 and 1-and-300.



(a) Mean estimates, random cross traffic model (b) Standard deviation estimates, random cross traffic model (c) Comparison of mean estimates between two cross traffic models

Figure 4: Comparisons of estimation results using different injection schemes and cross traffic models.

For static scheme, we use 1-and-100. The setting of the static scheme is for the worst link utilization case at the bottleneck link while that of the adaptive scheme does not care about the case. RLI receiver only produces per-flow latency estimates of regular traffic.

4.2 Results

We summarize our results with four aspects: accuracy of per-flow latency mean estimation, accuracy of standard deviation estimation, accuracy under bursty cross traffic model, and impact on loss of regular packets. In our experiments, with given regular traffic workload and queueing parameters, we observe about 22% link utilization, which always triggers the highest injection rate (1-and-10) in the adaptive scheme. Thus, note that the injection rate of adaptive scheme is always ten times higher than the static scheme in our experiments. We compare these two schemes with focus on the performance of the architecture with the above three metrics and evaluate if adaptive scheme with inaccurate link utilization can still works even across multiple hops.

Accuracy of latency mean estimation. In this experiment, we analyze how much accuracy is obtained by adaptive and static scheme as the link utilization of a bottleneck link (Switch2) increases under random cross traffic model. In Figure 4(a), we first observe that as the utilization increases, the accuracy of per-flow mean delay estimates also increases. Note that the percentage in the legend denotes link utilization at the second switch. For instance, in the static scheme, 70% of flows have less than 10% relative errors at 93% link utilization. Not surprisingly, adaptive scheme obtains better accuracy than static one due to its higher reference packet injection rate. Median relative errors at 67% and 93% utilization in static scheme are about 4.2% and 31%, where the difference is slightly less than an order of magnitude. For adaptive scheme, the difference in median relative error between 67% and 93% utilization is more significant than that by static scheme. The results show that higher injection rate is more effective to estimate per-flow latency accurately even in the presence of cross traffic.

Lower accuracy at 67% utilization is because actual per-flow latencies at 67% utilization are far smaller than those at 93% utilization. For instance, we observed the average latencies as 3.0μs and 83μs approximately, for 67% and 93% link utilization respectively. Thus, the lower accuracy at lower link utilization causes no significant absolute errors.

Accuracy of latency standard deviation estimation. We investigate per-flow latency standard deviation estimates under the same random cross traffic model. We observe a similar trend with mean estimates. Specifically, in adaptive scheme, while less than 10% relative error is obtained by about 30% flows at 67% link utilization, the same relative error is obtained by about 90% flows at 93% link utilization. Again, from the curves of adaptive scheme, median relative errors differ by about an order of magnitude between 67% and 93% utilization.

Accuracy under bursty cross traffic model. We conduct experiments with bursty cross traffic model. We then compare bursty traffic model with random traffic model in terms of the accuracy of per-flow latency estimates under the same link utilization. We set injection duration as 10 seconds for bursty traffic model and packet selection probability as 15%, which gives us 34% link utilization at the second switch. In a similar fashion, we achieve 67% link utilization for both traffic models.

From Figure 4(c), we observe that bursty arrival of cross traffic increases the accuracy of estimates significantly. We observe about an order of magnitude decrease of median relative error between bursty (1% median relative error) and random model (10% median relative error) at 67% link utilization. This improvement in accuracy is supported by the fact that the true value of average latency is much higher for bursty model (117μs as opposed to 3.0μs for random one) at 67% link utilization.

These results show that RLI architecture can capture the influence of adversary cross traffic successfully if any

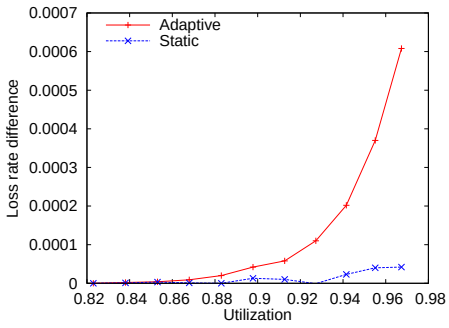


Figure 5: Reference packet interference.

and eventually help network operators isolate and diagnose problematic switches/routers across multiple hops.

Impact on packet loss. As we discussed earlier, adaptive scheme fails to adjust reference packet injection rate when a bottleneck link is not the one which an RLI sender is monitoring. As a result, the adaptive scheme produces reference packets at higher rate, which can alter the characteristics of traffic such as packet loss. Figure 5 shows packet loss increase (difference) caused by reference packets. In the figure, static scheme introduces extremely small perturbation with regular traffic in that there is at most 0.0042% increase in packet loss rate at about 97% link utilization. In case of adaptive scheme, packet loss rate difference increases up to 0.06%. While the increase by the adaptive scheme may look significant in comparison with the static scheme, 0.06% increase may still be within an acceptable range. More conservatively, we can use the static scheme considering worst case scenario since it yields reasonable per-flow latency estimation accuracy.

5 Related Work

A directly related work is mPlane [10]. mPlane addresses similar issues that we have for incremental deployment. The main difference between our scheme and mPlane is that our scheme deals with active per-flow latency measurement architecture but mPlane handles passive aggregate latency measurement solutions such as LDA [9].

A lot of prior work [4, 5, 15] exploited tomography techniques to infer hop and link characteristics from end-to-end measurements and topology information. They only provide aggregate measurements, not per-flow ones.

Duffield *et al.* proposed trajectory sampling for collecting packet trajectories across a network in [7]. Using these trajectory samples to infer loss and delay at different measurement points has been proposed in [16, 6]. Incorporating flow key in trajectory samples also enables per-flow latency estimation, and its efficacy was studied by Lee *et al.* in [12] while comparing their approach. In [12], the two timestamps already stored on a per-flow basis within NetFlow were exploited to obtain a crude estimator called Multiflow estimator.

Kompella *et al.* in [9] proposed a data structure called LDA. While LDA enables high-fidelity low network latency measurements in data center and algorithmic trading networks, it only provides aggregate measurements. RLI [11] exploits delay locality property and basic discussion is provided in Section 2. While RLI provides per-flow measurements, full deployment is costly.

6 Conclusion

Full deployment of active one-way per-flow latency measurement solution such as RLI in a data center network is most desirable to correctly localize latency-related problems in a switch or router. To reduce deployment complexity and support incremental deployment, we proposed a partial placement method called RLIR and evaluated the efficacy of the method in the presence of cross traffic across routers through simulation. The simulation results confirm that RLIR architecture supports partial deployment and works accurately without losing latency anomaly localization granularity significantly.

Acknowledgments

The authors thank the anonymous reviewers for comments on previous versions of this manuscript. This work was supported in part by NSF Awards 0831647 and 1054788, and a grant from Cisco Systems.

References

[1] Google Instant. http://www.google.com/instant.
[2] YAF: Yet Another Flowmeter. http://tools.netsa.cert.org/yaf/.
[3] 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.
[4] Y. Chen, D. Bindel, H. Song, and R. H. Katz. An Algebraic Approach to Practical and Scalable Overlay Network Monitoring. In *ACM SIGCOMM*, 2004.
[5] N. Duffield. Simple network performance tomography. In *ACM SIGCOMM Conference on Internet Measurement*, 2003.
[6] N. Duffield, A. Gerber, and M. Grossglauser. Trajectory engine: A backend for trajectory sampling. In *IEEE NOMS*, 2002.
[7] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *IEEE/ACM Transactions on Networking*, 2000.
[8] 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*, pages 98–105, 2002.
[9] 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.
[10] R. R. Kompella, A. C. Snoeren, and G. Varghese. mPlane: An architecture for scalable fault localization. In *ACM ReARCH*, 2009.
[11] M. Lee, N. Duffield, and R. R. Kompella. Not All Microseconds are Equal: Fine-Grained Per-Flow Measurements with Reference Latency Interpolation. In *ACM SIGCOMM*, 2010.
[12] M. Lee, N. Duffield, and R. R. Kompella. Two Samples are Enough: Opportunistic Flow-level latency estimation using Netflow. In *IEEE Infocom*, 2010.
[13] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *ACM SIGCOMM*, 2000.
[14] C. Shannon, E. Aben, kc claffy, and D. E. Andersen. CAIDA Anonymized 2008 Internet Traces Dataset (collection).
[15] Y. Zhao, Y. Chen, and D. Bindel. Towards unbiased end-to-end network diagnosis. In *ACM SIGCOMM*, 2006.
[16] T. Zseby, S. Zander, and G. Carle. Evaluation of building blocks for passive one-way-delay measurements. In *Proceedings of Passive and Active Measurement Workshop (PAM 2001)*, 2001.