

Two Samples are Enough: Opportunistic Flow-level Latency Estimation using NetFlow

Myungjin Lee[†], Nick Duffield[‡], Ramana Rao Kompella[†]

[†]Purdue University, [‡]AT&T Labs–Research

Abstract—The inherent support in routers (SNMP counters or NetFlow) is not sufficient to diagnose performance problems in IP networks, especially for flow-specific problems where the aggregate behavior within a router appears normal. To address this problem, in this paper, we propose a Consistent NetFlow (CNF) architecture for measuring per-flow performance measurements within routers. CNF utilizes existing NetFlow architecture that already reports the first and last timestamps per-flow, and proposes hash-based sampling to ensure that two adjacent routers record the same flows. We devise a novel Multiflow estimator that approximates the intermediate delay samples from other background flows to improve the per-flow latency estimates significantly compared to the naive estimator that only uses actual flow samples. In our experiments using real backbone traces and realistic delay models, we show that the Multiflow estimator is accurate with a median relative error of less than 20% for flows of size greater than 100 packets. We also show that prior approach based on trajectory sampling performs about 2-3x worse.

I. INTRODUCTION

Although IP networks were designed to be best-effort, an increasing number of applications today require performance guarantees. These range from multimedia applications such as voice-over-IP, video, multi-player online gaming, to performance-critical enterprise applications such as online trading. ISPs often provide SLA guarantees to ensure that the performance of their network matches up with the expectations of their customers that want to run these applications. SLA violations typically have heavy financial implications for the ISP; ISPs, therefore, care deeply about ensuring the health of their network.

IP networks, however, are notoriously hard to debug today. Apart from SNMP counters [1] such as for interface packet drops, there exists very little support in the routers for diagnosing performance problems. For example, routers today do not report aggregate performance statistics (*e.g.*, average delay, jitter, loss rate) of the forwarding paths (from one interface to another). Such performance statistics would be extremely useful for operators to diagnose the root cause of end-to-end problems, and take necessary steps to fix it. Without these intrinsic measurement capabilities, network operators today are forced to use external means to infer such statistics. For instance, ISPs today routinely monitor their networks with the help of measurement boxes that inject ‘active probes’ between routers. From the path properties observed by these probes, operators typically use inference algorithms to isolate the location of the problem.

Several inference algorithms based on tomographic approaches exist in the literature today (*e.g.*, [2], [3], [4]).

As noted before, however, the problem is generally under-constrained in this sense that, depending on topology on disposition of measurement points, not all individual link performances are visible. Another major problem is that they only provide aggregate statistics, not on a per-flow basis, which may be crucial to diagnose and debug flow-specific problems in the network. For example, a flow may have passed a router exactly when the busy period started for the router’s queue, because of which that particular flow may have obtained terrible throughput. When aggregated over a larger interval, it may appear that the forwarding path within the router is functioning correctly and thus may not be detected using active probes alone.

Thus, despite their limitations in coverage and granularity, network operators still rely on active probes as they do not have any other alternative today. From the router vendors’ point of view, routers are already burdened with the ever increasing suite of protocols they need to support and extremely high line rates, making it quite challenging to provide these measurement capabilities. From the network operator’s perspective, obtaining such measurements directly from the routers would definitely make the task of debugging their network easier. As a first attempt to bridging this gap between the capabilities of routers and the ISP needs, in this paper, we consider the problem of obtaining hop-level latency measurements on a per-flow basis.

At first glance, this goal may appear challenging to achieve. We need precise time-synchronization, which has been traditionally hard, as well as coordination between routers, which is often met with resistance since routers are already over-burdened. The ground has, however, significantly shifted in terms of the capabilities of routers today. For instance, routers are being equipped with sophisticated time-synchronization primitives such as the IEEE 1588 protocol [5], GPS-based clocks, etc. There have been significant innovations in designing router primitives that facilitate coordination between routers without explicit communication [6].

To describe our architecture for obtaining per-flow latencies, we start with sampled NetFlow as our main point of departure. There are two key ideas: The first idea is to exploit the fact that NetFlow already maintains two timestamps corresponding to the first and last packets of a flow. We ensure that two NetFlow processes running at two different routers (or more generically any segment along an end-to-end path) maintain the same flows and timestamps for the same first and last packets. This *Consistent NetFlow* architecture enables us to obtain two delay

samples for the flow. We can easily achieve this using hash-based packet sampling. The second idea is to opportunistically refine the latency estimates of a flow by utilizing the delay samples obtained from other flows that lie within the flow’s duration. We show that how our novel *Multiflow estimator* based on this idea provides significantly better estimates of per-flow latencies compared to the Endpoint estimator that uses the two delay samples obtained using consistent NetFlow.

There is one main challenge that remains in the above architecture. Our architecture provides per-flow latency estimates for only the sampled flows and, not *all* flows. We believe this problem is fundamentally due to the existing router resource constraints that any passive measurement solution has to deal with. As technology improves, so will the router resources to support higher sampling rates; the fact that our architecture lies within the NetFlow framework makes this scaling automatic and straightforward. While the fact that our architecture can provide reasonable estimates of per-flow latencies within routers is significant, it is not intended to serve as a replacement for sophisticated measurement frameworks (*e.g.*, GPS-enabled setup used in [7]). Instead, we envision our approach to provide low-cost ways of continuously monitoring latency characteristics of flows across routers. Thus, when any SLA violations are observed in the network, our architecture will allow easy localization of the router or interface at which such a violation occurred.

Thus, the *main contributions* of this paper include the design of Consistent NetFlow (discussed in Section III-A) and an opportunistic latency estimator called the Multiflow estimator (discussed in Section III-C) that provides per-flow latency estimates on a per-hop basis. The Multiflow estimator’s accuracy empirically using real backbone traces under different delay models. Our results indicate that the Multiflow estimator provides a median relative error of less than 20% for flows greater than 100 packets. In contrast, we observe that trajectory sampling (when adapted for this purpose) is 2-3 times worse than our Multiflow estimator. We describe the experiment details and results in Section IV.

II. RELATED WORK

NetFlow [8] is the main basis for our approach. Several variants of NetFlow (*e.g.*, Adaptive NetFlow [9], Flow slices [10]) and other passive measurement data structures [11], [12] exist, but they only measure flow characteristics such as number of packets, bytes, and flags. To the best of our knowledge, we are the first to propose integrating latency measurements into the NetFlow framework, and a concrete architecture and estimators to achieve such an integration.

Tomography approaches are standard for predicting the average latency characteristics of links and router forwarding paths [13], [3]. They solve the problem of predicting the per-hop loss and latency characteristics based on end-to-end measurements (*e.g.*, conducted using active probing tools [14], [15]) and routing information obtained from the network. Our work differs from theirs since we use passive measurements while they rely on active probes.

Our work shares some similarity with trajectory sampling in [6]. Specifically, we borrow consistent hashing in our effort for ensuring that consistent streams of packets are observed at two routers. The notion of a flow plays a fundamental role in our work (and therefore the NetFlow collection framework) while trajectory sampling relies on flat packet labels with no flow-level aggregation. Passive measurement of loss and delay by directly comparing trajectory samples observed at different points has been studied [16], [17]. Although our proposal is targeted toward the aggregate reporting paradigm on NetFlow, our approach could potentially also be used to augment collector side analysis in packet-level passive delay measurement of the work in [16], [17]. While we do not compare our proposal with [16] because [16] mostly focuses on evaluating the performances of several schemes making packet digest for packet-level delay measurement, we compare ours with trajectory sampling [17] in the context of per-flow delay estimates in the Section IV.

There have been other prior approaches that attempt to obtain direct performance measurements from routers. For example, Machiraju *et al.* suggest a measurement-friendly network (MFN) architecture as a router primitive which uses hop-dependent priority queuing [18]. This architecture requires a lot of intrusive changes in routers, however. It also does not provide per-flow latency estimates. A very recent work by Kompella *et al.* in [19] propose a high-speed latency detection data structure called lossy difference aggregator (LDA). LDA requires new data structures in routers and hence may not be deployed easily.

III. FLOW COLLECTION ARCHITECTURE

In this section, we discuss our *Consistent NetFlow* architecture for supporting delay measurements, and pin-point the changes required in current NetFlow to implement it.

A. Consistent NetFlow

In today’s flow collection architecture, individual routers in a network run NetFlow [8] on various interfaces (typically on the ingress direction). Given that backbone routers cannot keep up with high line rates (*e.g.*, OC-192 or 10 Gbps), routers typically run a variant called Sampled NetFlow that uses a simple stage of uniform packet sampling (rates from 0.001 to 0.01) to ensure that the processor as well as the memory resources are not overwhelmed. The line-card CPU computes individual flow record by aggregating all packets with same *flow key* in the sampled packet stream. Typically, the flow key comprises the TCP 6-tuple consisting of the source and destination IP addresses and ports, protocol and interface number. NetFlow also records the timestamps of the first and last packets observed for that flow.

Our goal is to retrofit delay measurements into the NetFlow architecture described above. Without loss of generality, let us consider measuring the delay between NetFlow instances *A* and *B* in Figure 1; this will essentially measure forwarding path latencies in *R1* from input port *A* to output port connected to *R3*, and the link delay between *R1* and *R3*. We start with

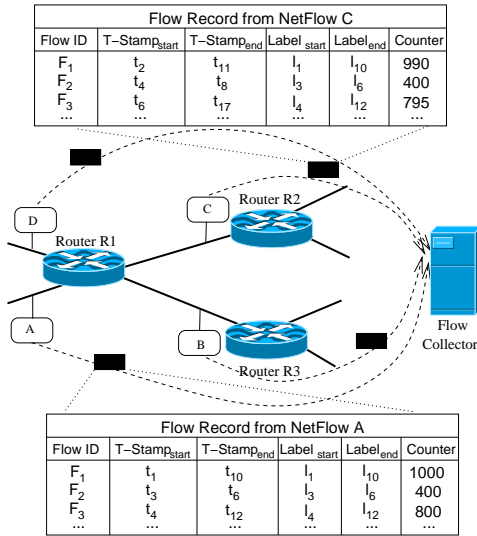


Fig. 1. Overview of our Consistent NetFlow architecture. The main change compared to the current NetFlow architecture is the hash-based sampling stage shown in the NetFlow process. The flow collector aggregates flow records from individual routers and extracts a time-series of delay samples from which per-flow latency measurements are recorded.

the observation that NetFlow instances *A* and *B* already store the timestamps of the first and last packets for the set of sampled flows. The *key idea* is that, if both *A* and *B* sample the same first and last packets (p_0 and p_n) for some flow recorded by both (in their set of sampled flows), then we can trivially compute the delay observed by these packets p_0 and p_n from the timestamps recorded at both *A* and *B*. Of course, we need to assume that both *A* and *B* are time-synchronized for the measurements to be accurate.

In the original NetFlow architecture, both *A* and *B* sample packets completely independently, however. In order to address this problem, we propose a *Consistent NetFlow* (CNF) architecture, that is essentially NetFlow with the additional modification that routers use *hash-based sampling*. Thus, in CNF, routers independently select the same packet set and thus timestamps are consistent across NetFlow instances.

Hash-based sampling. Each router calculates a hash over some set of packet fields that are invariant over the packet path, and the packet is selected for reporting if the hash falls into a given range. When all measuring routers use the same hash function, input fields, and selection range, their selection is consistent. Hash-based sampling was proposed in [6], and has now been standardized in the IETF [20], [21]. With a suitably strong hash function, the average sampling rate can be determined as the size of the selection range divided by the size of the set of possible hash values. Note that the standards are flexible over what fields are used as input to the hash. By changing an input ranging from invariant fields of flow key to packet payload, we can achieve *flow sampling* [22] or select packets consistently between different routers (unless otherwise mentioned, we refer to this form of sampling henceforth as just *packet sampling*).

While in theory, both sampling mechanisms can be used

to ensure that same flows are recorded, hash-based packet sampling is often preferred because flow sampling can lead to bursts while packet sampling results in smoother selection of packets. In addition, packet sampling is also famously biased towards large flows, for which our goal of identifying latency measurements may be even more critical. Typically, there is an additional level of sampling during flow export to a flow collector to reduce the reporting bandwidth. For simplicity, however, we assume all sampled flows are transmitted but our architecture works even if such sampling were performed.

Time synchronization. In addition to the above, we require accurate time synchronization between the end-points. This is a fundamental requirement for *any* architecture that wishes to enable accurate delay measurements. The Global Positioning System (GPS) is used to synchronize clocks at different measurement points; see *e.g.*, [7]. Router vendors are also increasingly considering hardware protocols such as IEEE 1588 [5] to synchronize routers within μ second precision. Finally, one implicit assumption is that the stream of packets follows a serial order (FIFO) between the monitoring points. While this is mostly true, some routers can employ fair queuing mechanisms, and thus, packets may pass through separate forwarding paths within a router. In such cases, we require that different forwarding paths be treated differently.

B. Flow Correlation

Expired flow records are transmitted by the NetFlow process on each of the routers to a centralized flow collector as shown in Figure 1. Flow records obtained at any given NetFlow instance (say *A*) will contain flow records that would have potentially taken different paths at the router (to *B* and *C*). Thus, we need the *intersection set* of the flow records obtained from the end-points (between *A* and *B*, *A* and *C*). The second step is to associate these sets of flow records with each other. By construction, each flow record at the downstream instance *C* will have a corresponding entry at the upstream instance *A*, but there could be multiple flow records with the same flow key due to flow expiration timeouts in NetFlow. In many cases, we can trivially associate by sorting and pair-wise associating flow records using the start timestamps. There could still be cases, however, due to packet losses, or clock synchronization. One way to eliminate this is to use a packet label (using the hash digest of a packet), but that would require non-standard modifications. We outline a mechanism based on just timing checks to eliminate such inconsistencies.

Timing checks to eliminate inconsistencies. The boundary between flows reported at the sender and receiver side may be different due to (i) packet loss, (ii) differences in application of flow timeouts due to slight difference in inter-packet times, (iii) different cache expiry events due to heavy loads, and (iv) uncertainties in timestamps due to propagation delay and clock synchronization issues. The following scheme addresses these problems. We can treat a packet stream with a single key, since flows with different keys at sender and receiver are not matched. The sender exports a set of flow records i with fields $(\tau_{si,1}, \tau_{si,2}, n_{si}, b_{si})$ being the initial, final packet timestamps

number of packets and bytes respectively. The receiver side flows have corresponding fields $(\tau_{ri,1}, \tau_{ri,2}, n_{ri}, b_{ri})$. But note the same index i on sender and receiver side is not assumed to originate in the same set of packets.

We now consider four different quantities e_1^-, e_1^+, e_2, e_3 that represent timing uncertainty:

- The propagation delay lies in the interval $[e_1^-, e_1^+]$.
- The packet processing latency is less than e_2 .
- The clock uncertainty is less than e_3 .

Let T_{in} and T_{act} denote the inactive and active flow timeouts respectively. We consider the following criteria (C1)–(C5) applied to matching sender flow i and receiver flow j . In practice, these are applied as follows. We seek pairs of sender and receiver flows that are time compatible, *i.e.*, they satisfy (C1). Since flows of a given key can be ordered by first packet time, this requires only a windowed search. Each (C1) compatible pair is then examined to check they satisfy (C2)–(C4). If so, they can be passed to the analysis stage and we return to seek another (C1) compatible pair. If the flows fail any of the conditions (C2)–(C5), they are discarded.

- (C1) $\tau_{ri,1} \in \tau_{sj,1} + [e_1^- - e_3, e_1^+ + e_2 + e_3]$ and $\tau_{ri,2} \in \tau_{sj,2} + [e_1^- - e_3, e_1^+ + e_2 + e_3]$. This condition says the first and last packet times of receiver flow i are compatible with first and last packet times of sender flow j , given the uncertainties of propagation, queuing, and clock synchronization.
- (C2) $n_{ri} = n_{sj}$ and $b_{ri} = b_{sj}$. Equality of bytes and packets is a necessary condition for the flows to be reporting on the same set of packets.
- (C3) $\tau_{ri,2} - e_1^- + e_3 < \tau_{sj,1} + T_{act}$. The upper bound for latest possible compatible sending time of the last flow packet seen at the receiver $\tau_{ri,2} - e_1^- + e_3$, should fall within the active timeout since the first sender packet.
- (C4) $\tau_{ri,2} - e_1^- + e_3 < \tau_{sj,2} + T_{in}$. This condition is similar to (C3) except that it rules out sender packets being excluded from the flow due to inactive timeout since the last sender packet.
- (C5) If a pair of sender and receiver flows (of a particular key) has been discarded, discard all subsequent sender and receiver flows (of the same key) until the separation between successive flows exceeds T_{in} on both sender and receiver sides. This is used because after discard of a pair of flows for violating any of conditions (C2)–(C4), the first times of subsequent flows may refer to different packet on the sender and receiver sides. The condition ensures discard of such “out of sync” flows.

C. Latency estimation

Once pairs of flow records are associated, we have sender and receiver timestamps from both its first and last packets (but for just one packet in the case of a single packet flow). Using these packet timestamps, our goal is to estimate per-flow latency characteristics of the particular segment. We consider several estimators in increasing degree of accuracy. Before we discuss our estimators, we briefly outline the basic intuition behind our estimators.

Foundation: Delay Correlation. The central premise behind our approach is that when two packets traverse a link closely separated in time, then the queuing delays that they experience are positively correlated. We then draw the conclusion that it makes sense to estimate the delay of a packet that we cannot measure directly, from the delays of closely separated packets that we can measure directly.

At an intuitive level, packets that experience the same queuing busy periods should tend to have positively correlated delays. There are a number of analytical results that confirm this behavior for classes of Markovian queuing models, such M/G/1 or G/M/1. For our purposes, there are informative results for models of this nature. In the simplest cases, the lag n autocorrelation of the packet queuing delay, γ_n , obeys

$$\gamma_n = 1 - n(1 - \rho)^2 + O(1 - \rho)^3 \quad (1)$$

where $\rho \leq 1$ is the offered load; see [23]. This formula shows that under heavy traffic conditions (ρ close to 1), noticeable correlations persist at lag n for n up to $1/(1 - \rho)^2$, *e.g.*, up to $n = 100$ for a 90% load.

Although this result applies to a very simple model, there are two reasons to suggest that correlations in packet delays at a queue whose input is typical Internet traffic will be even greater. First, under more general conditions than (1) was derived, γ_n is a convex function of n , and hence the correlations in practice decay more slowly than the dominant linear behavior in n of (1). Second, Internet traffic exhibits burstier arrivals than the simple models [24]. So, for a given load, packets that queue are more likely to do so behind other queuing packets, and consequently waiting time autocorrelation will be greater than in the Markovian case.

The power of this idea for delay measurement is that, we can enhance the packet delay estimates of a particular flow, with directly measured delays from nearby packets from potentially other flows, in particular, arriving between the first and last packets of the flow under study.

Endpoint Estimator. The Endpoint method is the naive estimator that averages the pair $\Delta_{EP} = \{d_1, d_2\}$ of the delays of the first and last packets of the flow:

$$D_{EP} = \frac{d_1 + d_2}{2}$$

Multiflow Estimator. Although the Endpoint estimator is exact for flows having two packets, it is expected to have limited accuracy in general precisely because it is based only on two samples. We propose the Multiflow estimator based on the autocorrelation description above. We first construct the matching streams τ_{si} and τ_{ri} , $i = 1, 2, \dots$ of matching sender and receiver timestamps of all first and last packets of sampled flows. The Multiflow estimator does not distinguish between first and last packet timestamps. Now consider a particular flow under study with sender timestamps $\tau_{si,1}$ and $\tau_{si,2}$ and receiver timestamps $\tau_{ri,1}$ and $\tau_{ri,2}$ where $i, 1$ and $i, 2$ are the indices of the first and last packets of the flow. We form a delay estimate by averaging the delay over all packets with sender timestamps between the sender timestamps of the flow.

Specifically, for $r_1 < r_2$ let $S(r_1, r_2) = \{i : r_1 \leq \tau_{si} \leq r_2\}$. The set of packet delays associated with the flow is

$$\Delta_{MF} = \{\tau_{ri} - \tau_{si} : i \in S(\tau_{si,1}, \tau_{si,2})\}$$

and the Multiflow estimator is

$$D_{MF} = \frac{\sum_{\delta \in \Delta_{MF}} \delta}{|\Delta_{MF}|}$$

where δ is a delay sample in the set Δ_{MF} .

Hybrid estimator. The hybrid method attempts to blend the best of the Endpoint and Multiflow methods. In our experiments in Section IV-D we shall find that the Endpoint method tends to be more accurate for smaller flows; indeed in the special case of a two packet flow without loss, it is exact. In the hybrid approach we apply a threshold in the number of packets per flow, using the Endpoint method for smaller flows and the Multiflow method for larger flows.

D. Variance and its Estimation

In measuring delay, it is useful to produce not just a single summary statistic, the mean, but also a measure of the variability of the distribution of individual delays about that value. Thus, we ask two questions: (i) given the expected correlations between delay measurements, how well is the variability of a single measurement predicted? (ii) how does the variability of set of samples Δ relate to that of the packets in the flow under study itself?

To answer the first question, if the delay samples $\delta \in \Delta$ were i.i.d random variables, we would form the unbiased estimate of the population variance as

$$\sigma^2 = \frac{\sum_{\delta \in \Delta} (\delta - D)^2}{|\Delta| - 1}$$

where $D = (|\Delta|)^{-1} \sum_{\delta \in \Delta} \delta$ is the sample mean. In the presence of positive correlations between the δ , namely $E[\delta\delta'] > E[\delta]E[\delta']$ for δ, δ' distinct elements of Δ , then $E[\sigma^2] > \text{Var}(\delta)$. Thus we would tend to overestimate the variance, which can be regarded as conservative for performance applications. We will answer the second question through experiments in Section IV. For the moment we make the observation that for the Endpoint estimator, the quantity σ^2 is expected to be extremely noisy, being based on only two data points; this is indeed the case found in Section IV.

E. Interpolation of Packet Delays

Given our motivating intuition concerning correlation of packet delays, the correlation between the delay of arbitrary packet in a flow, and the endpoint delay of another flow should be strongest if we could choose the endpoint to have closest possible sender time to the packet in question. Unfortunately, we cannot construct an estimator this way, since we do not know the sender times of individual packets. On the other hand, for a performance study, choosing to estimate with a close-by packet indicates the limits of accuracy of any method, such as our Multiflow estimator, which uses delays of other flows as estimates.

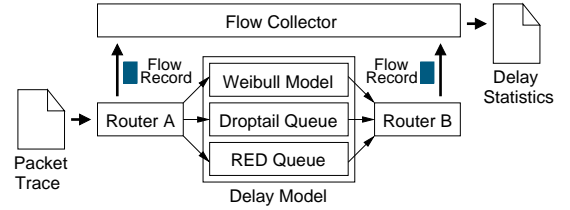


Fig. 2. Simulation environment.

To this end, we use linear interpolation between endpoint delays in order to construct a delay value for any time, then compare it with individual packet delays (not just endpoints) of arbitrary flows. Let $d_i = \tau_{ri} - \tau_{si}$, $i = 1, 2, \dots, n$ denote the set of flow endpoint packet delays, ordered with increasing send time τ_{si} . For any time x between τ_1 and τ_m , let $i_+(x)$ and $i_-(x)$ label the closest delay values in the future and past:

$$i_+(x) = \min\{j : \tau_j > x\} \text{ and } i_-(x) = \max\{j : \tau_j < x\}$$

Then the interpolated delay at an arbitrary time x between τ_1 and τ_m is

$$d_{\text{int}}(x) = d_{i_-(x)} + (x - \tau_{si_-(x)}) \frac{d_{i_+(x)} - d_{i_-(x)}}{\tau_{si_+(x)} - \tau_{si_-(x)}}$$

For a lossless flow of m packets, having sets of $\{\tau_{si} : i = 1, \dots, m\}$ and $\{\tau_{ri} : i = 1, \dots, m\}$ of packet send and receive times, we construct the set of interpolated delay values for interior packets, and actual endpoint delays:

$$\Delta_{\text{int}} = \{\tau_{r1} - \tau_{s1}, \tau_{rm} - \tau_{sm}\} \cup \{d_{\text{int}}(\tau_{si}) : i = 2, \dots, m-1\}$$

which we compare with the actual packet delay values $\Delta_0 = \{\tau_{ri} - \tau_{si} : i = 1, \dots, m\}$.

IV. EVALUATION

In this section, we evaluate the efficacy of our Multiflow estimator in obtaining accurate per-flow statistics using real backbone packet header traces and different delay models. We also consider the impact of several variables such as packet loss rate, packet sampling rate, etc. on the accuracy of our estimates. Before we describe the results, we first outline our experimental setup.

A. Experimental Setup

There are three main components in our experimental setup—packet header trace, delay model, and flow collection tool as shown in Figure 2. For our experiments, we need correlated traces at two different routers with per-packet timestamps. There are *no* such publicly available traces. Pappagiannaki *et al.*, however, have collected such traces from a backbone router in [7] to create a model for the delay distributions. Thus, we resorted to using a general-purpose packet header trace obtained from a single monitoring point, and simulated different delay distributions. The flow collector allows us to take packet header traces and form flow records just as NetFlow would. Each of the components are explained next in detail.

Traces. We use two real backbone packet header traces. Both traces contain no payload information, and all IP addresses are anonymized. The CHIC trace is collected by a monitor located at Equinix in Chicago, IL, contains traffic for 60 seconds from 5:00 PM, April 30th, 2008 on an OC-192 backbone link published by CAIDA [25]. This trace has about 1 million flows and 13 million packets in 60 seconds. The IPLS trace published by NLANR [26] contains traffic between Indianapolis and Kansas City through an OC-48 link (2.5 Gbps) collected on August 14th, 2002 at 9:20AM in the Abilene network. There are approximately 0.8 million flows and 17 million packets in this trace.

Flow collection tool. We used an open-source NetFlow platform called YAF [27] for our evaluation. We modified YAF to support hash-based packet sampling and flow sampling. In addition, we also extended YAF to report the hash label for the first and last packets of a flow for comparing the efficacy of our timing-based checks for packet association.

Delay models. As shown in Figure 2, we have implemented three different delay models. The first is a Weibull distribution that Papagiannaki *et al.* have empirically found to model packet delays in real backbone routers [7]. The other two models are based on simulating queuing dynamics according to the Droptail and RED queue management [28] strategies. For the Weibull distribution, we set the scale and shape parameters to 0.0001 and 0.6 (as recommended in [7]) for both CHIC and IPLS traces. We model packet losses as a uniform distribution for this delay model. For the queuing models, we control the packet delays by configuring queue length and drain rate. We set the queue length to 10,000 for CHIC trace and 3,000 for IPLS trace.

While Droptail requires no further parameters, RED needs configuring the queue weight (w_q), minimum and maximum thresholds (min_{th} and max_{th}), and maximum drop probability (max_p). Following the guidelines in [28], we chose a queue size of 10,000, $min_{th} = 4,000$ and $max_{th} = 9,000$ for the CHIC trace. For IPLS trace, we chose a queue size of 3,000, $min_{th} = 1,000$ and $max_{th} = 2,500$. We use $w_q = 0.002$ and $max_p = \frac{1}{50}$ for both traces.

B. Associating flow records

We first compare the efficacy of the timing-based approach by running the constraint checks (outlined in Section III-B) over the CHIC trace. We set $e_1^- = 10ms$ and $e_1^+ = 20ms$. We set e_2 by multiplying maximum queue size by average packet processing time (which varies from 40ms to 47.6ms respectively). We set the clock uncertainty parameter e_3 to 1ms to ensure worst case. We set $T_{in} = 10s$ and $T_{act} = 30s$ which is unusually low for T_{act} (default on Cisco routers is 30 minutes). We chose this low value to test C3 given the trace is only 60 seconds long.

Table I shows the percentage of flows obtained by filtering flows that fail various constraints outlined in Section III-B. At no packet loss, the packet label approach results in all flows included, while at about 0.98% loss, we lose approximately 0.76% of flows because the packet labels do not match. The

(Pkt. sampling rate = 0.005, Total flows = 40,564)

Pkt. Loss	C1	C2	C3	C4	Pkt. Label
0.00%	100.00%	100.00%	99.99%	99.99%	100.00%
0.98%	98.77%	98.44%	98.43%	98.43%	99.24%
2.11%	97.21%	96.57%	96.56%	96.56%	98.20%
2.91%	96.24%	95.38%	95.37%	95.37%	97.64%
3.92%	94.88%	93.85%	93.84%	93.84%	96.65%
4.93%	93.73%	92.42%	92.41%	92.41%	95.85%

TABLE I
TIMING CHECKS COMPARED TO PACKET LABEL APPROACH.

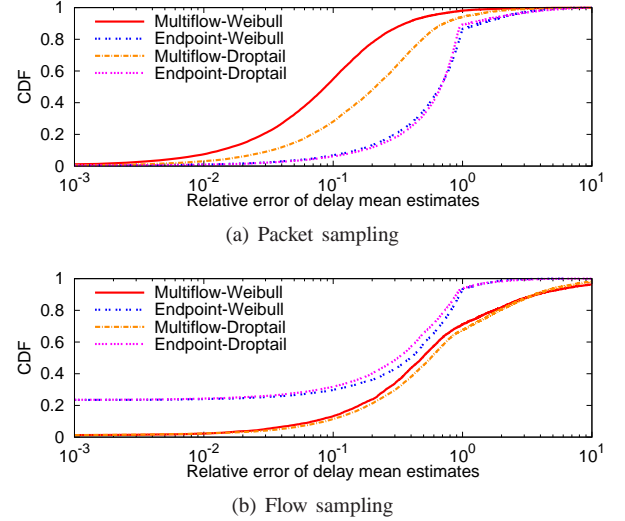


Fig. 3. Accuracy of estimators using Weibull distribution model on CHIC trace for flow and packet sampling. Flow and packet sampling rate = 0.01.

table also indicates that C3 and C4 result in obtaining the same set of flows since there were no instances of C4 in the trace (as discussed in Section III-B). We found in all cases, that the set of flows obtained by applying the constraints are always a proper subset of those obtained using the packet labels. Due to simplicity of the packet-label approach, we used label-based association in our experiments, but we removed top 1% largest delays to ensure no abnormal delay samples affect our results.

C. Estimator accuracy

We first provide basic results outlining the efficacy of our estimators. We simulate both packet as well as flow sampling approaches. For simplicity, we also assume no packet loss for these cases; we discuss packet loss variation later in Section IV-D. In Figure 3(a), we plot the CDF of the relative error of mean delay estimates for the Multiflow and Endpoint estimators. We show the curves for both Weibull distribution as well as Droptail queuing model (RED is exactly the same since there is no packet loss). We can observe that the Multiflow method obtains a median relative error of 10% and an 80th percentile relative error of about 20% for the Weibull delay model. The accuracy is slightly lower for the Droptail queuing model. The Endpoint estimator however is significantly inaccurate (a median error of 50%) compared to the Multiflow estimator for both distributions.

Figure 3(b) shows a similar comparison between Endpoint and Multiflow for flow sampling. Curiously, Endpoint appears to perform better than Multiflow. The fundamental difference

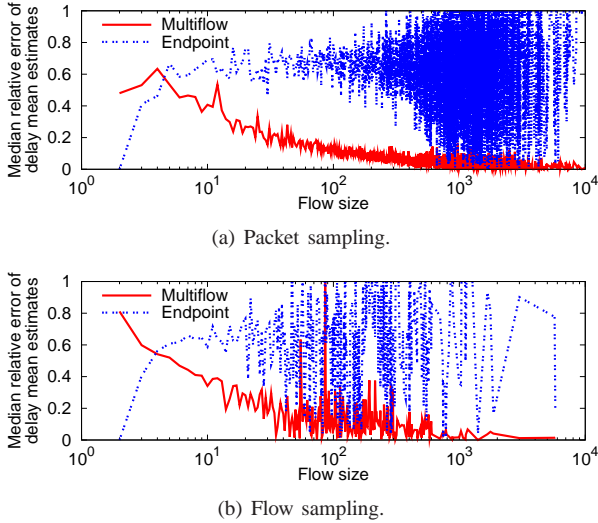


Fig. 4. Median relative error of delay mean estimates depending on flow size. Weibull distribution is applied to CHIC trace.

between flow sampling and packet sampling is the fact that flow sampling uniformly samples flows while packet sampling is known to be biased towards heavy-hitters, as large flows get sampled more often. So, in order to understand this deeper, we study the relationship between the estimator error and flow sizes (number of packets in a flow).

We plot in Figure 4 the variation of median relative error of mean delay estimates for different flow sizes. In other words, each point represents the median relative error among all flows that are of a particular size as defined by the x -axis. From the figures, we can clearly observe that the relative error of the Multiflow estimator decreases as the flow size increases for both flow- as well as packet-sampling. For flows of size greater than 200, the median relative error is less than 10% for packet sampling. In contrast, the error suffered by the Endpoint estimator increases as flow size increases significantly. More importantly, it becomes extremely erratic indicating that it is not a reliable predictor of flow latency estimates.

For both types of sampling, somewhat curiously, we can observe from Figures 4(a) and 4(b) that the Endpoint performs better than the Multiflow estimator for flows up to a size of 3-4 (or so) and then the accuracy decreases. While the figures are for Weibull distribution, we have observed similar patterns for RED and Droptail as well. We believe this is because typically for small flows, an estimate from two actual delay samples is better than approximating using all the intermediate packets. Thus, a hybrid estimator that combines the two approaches will provide strictly better accuracy than either of the two as we have discussed in Section III-C.

Now, we return back to the question as to why the Endpoint estimator appeared better than the Multiflow estimator. For smaller flow sizes (< 4 packets), the Endpoint estimator was more accurate than Multiflow. The fact that flow sampling contains a large number of small flows, meant that the overall distribution was heavily skewed towards these small flows. For flow sampling, the number of small flows is much larger

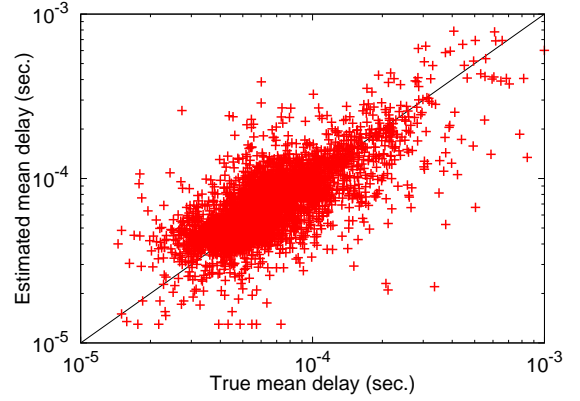


Fig. 5. Scatter plot showing the true as well as estimated delay spread across all flows.

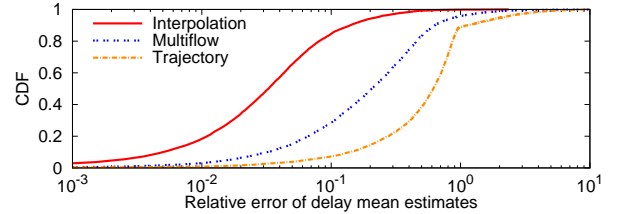


Fig. 6. Relative error for interpolated delay and Multiflow delay estimator for packet sampling. RED queuing model with no packet loss is used for CHIC trace.

than that using packet sampling (11,342 out of 11,721 for flow sampling as compared to only 5,436 small flows out of 16,178 for packet sampling). Thus, the distribution is dominated by the small flows for flow sampling.

Unbiasedness and delay-spread. To illustrate the unbiased nature of our Multiflow estimator, we show a scatter plot in Figure 5 with the true mean delay on the x -axis and the estimated mean delay on the y -axis for each and every flow. Two main conclusions can be drawn from the figure: First, when all the flows are considered, there is quite a bit of spread in the true latency characteristics of the flows ranging all the way from 10^{-5} to 10^{-3} seconds, although most of the delays lie between 2×10^{-5} and 10^{-4} . This in some sense motivates why we need estimators such as ours; if all average delays were the same, per-flow estimators would not have been required. Second, the Multiflow estimator appears to have two-sided errors indicating its unbiased nature.

Comparison with Interpolation and Trajectory sampling. In this experiment, we compare our Multiflow estimator with the *hypothetical* interpolation approach and a prior approach based on trajectory sampling [6]. Trajectory sampling can potentially be used to estimate per-flow average delays assuming that each packet label in trajectory sampling is augmented with extra flow information (the original proposal [6] was only to include flow information at a sample taken from the network ingress). For obtaining per-flow latency estimates in trajectory sampling, we group together the original set of packet samples based on the flow key and compute the estimates based on these samples.

We observe from Figure 6 that the relative errors obtained

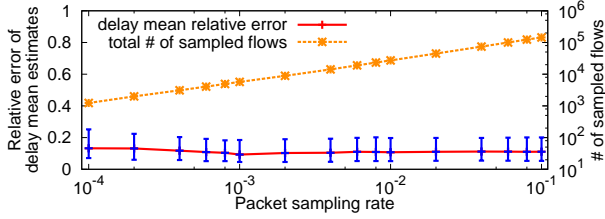


Fig. 7. Dependency of delay mean estimates on packet sampling rate. Each curve represents median with error bars indicating 25th and 75th percentile where flow size > 100 packets for IPLS trace.

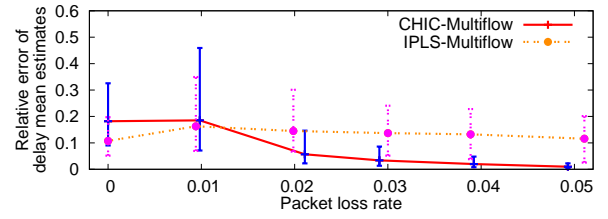
using our Multiflow estimator are much higher than those obtained via interpolation for the CHIC trace. The horizontal distance between the curves is nearly an order of magnitude: an error of a given likelihood is about 8 times as large for Multiflow as for the hypothetical interpolation approach. The horizontal distance from Multiflow to Trajectory Sampling is a little more than another half an order of magnitude, meaning an error of a given frequency is about 4 times larger for trajectory sampling as for Multiflow. For example, the median relative error (0.5 on the vertical axis) is about 0.03 for Interpolation, 0.2 for Multiflow, but about 0.6 for Trajectory Sampling. For the IPLS trace (omitted for the brevity), the curves are in the same order but tighter, with errors of a given likelihood being a factor of 2 greater for Multiflow as for Interpolation, with Trajectory Sampling another factor of about 3 greater. These results indicate that Multiflow has strikingly better accuracy than the Trajectory Sampling approach, but there is still plenty of room for improvement. We could likely achieve the accuracy of the interpolated delay estimator if more information about packet inter-arrival were provided.

D. Sampling and loss rate variation

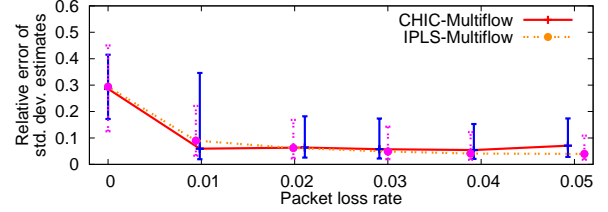
We have two variables that control the effective number of sampled packets—packet sampling rate and loss rate. Since the Endpoint estimator does not function as well as Multiflow, we only show the results for Multiflow. We also only consider packet sampling henceforth in the interest of space.

Impact of sampling rate. For understanding the relationship between estimation accuracy and packet sampling rate, we ran the experiments varying packet sampling rate from 0.0001 to 0.1 for both CHIC and IPLS traces using RED queue model. We only show the results of IPLS trace in this paper due to space limitation. We configured per-packet processing time to produce no packet loss in the experiments.

We show the influence of packet sampling rate on the estimator accuracy of large flows in Figure 7. We can observe that while relative errors decrease as the packet sampling rate increases, they stabilize after a sampling rate of about 10^{-3} for the IPLS trace and 10^{-2} for the CHIC trace (not included in the figure). While we do not show a similar graph for small flows (of size ≤ 100), we note that there is one major difference from the corresponding large flows' graph. We observe that as packet sampling rate increases, the relative errors for small flows also increase. We found this trend in both traces. While not as pronounced as packet sampling, we



(a) Mean



(b) Std. Deviation

Fig. 8. Relationship between mean and std. deviation of delay estimates and packet loss rate. The curves show median and the error bars indicate 25th and 75th percentiles. Flow size > 100 packets. RED queue model is used.

also observed this in flow sampling (omitted for brevity). We believe this phenomenon could be because of the fact that increasing packet sampling rate leads to an increase in delay samples. While we expect larger number of delay samples to be beneficial in general, the problem is that estimated delays now are aggregated over larger number of samples, which benefits larger flows but reduces smaller flows' accuracy. This observation is similar to that observed in Figure 3(b) where the Endpoint estimator was better than the Multiflow estimator.

Impact of packet loss rate. Higher loss rate typically reduces the effective number of samples from which we can compute the average per-flow latencies. We vary packet loss rate from 0% to 5% for both IPLS and CHIC traces by changing the drain rate of a queue. While 1% loss rate seems a reasonable maximum loss rate, we test up to 5% loss rate for understanding the impact of such a high packet loss rate. While we performed experiments for all three delay models, we mainly analyze results of RED while briefly summarizing other results. We show the RED results in Figure 8(a). Normally, one would expect that as loss rate increases, the error should increase as lesser number of samples exist. But, contrary to our intuition, in Figure 8(a), relative error significantly reduces as we increase the loss rate. The explanation for this is as follows. In RED or Droptail queues, loss rates and delay characteristics are inter-twined; high loss rate implies that the queue is almost always full. Thus, the delay distribution is a lot more stable and hence easier to predict even with smaller number of samples (as a result of the loss rate). While our results for a Droptail queue model were similar to RED, the Weibull delay model shows relatively constant error in the latency estimates despite increase of packet loss rate. This is due to the fact that packet losses are independent with delays in our settings.

E. Accuracy of standard deviation estimates

We now explore the accuracy of the standard deviation estimator outlined in Section III-D. Just as before, we compare the accuracy of both the Multiflow and Endpoint estimators.

Similar to the case of mean delay estimates, the increase of the packet loss rate reduces the relative error of standard deviation of flow-level latency. The main difference though is that, while in case of delay mean estimates there is a range of loss rates in which relative error increases, no such range exists for standard deviation. In Figure 8(b), we observe that the Multiflow estimator relative error for large flows decreases significantly. The same trend exists for both CHIC as well as IPLS traces. The Endpoint estimator on the other hand exhibits close to 100% error for all packet loss rate ranges. We omit the actual graph due to space limitation.

One curious observation is that, at around 4%-5% packet loss rate, the relative error of Multiflow on CHIC increases slightly. Upon careful investigation, we have observed that, as link utilization become higher (which happens when we increase the processing time and thus increase the loss rate), the variance of packet delay becomes very small, and thus the relative error of the standard deviation becomes large. For instance, in CHIC trace, true and estimated delay standard deviations of a flow were about 0.016 and 0.015 at 2.11% packet loss rate, of which relative error was 6.25%. However, for the same flow, true and estimated delay standard deviations were about 0.006 and 0.007 at 4.93% packet loss rate leading to a relative error of about 17%.

We also studied the dependence on flow size of the relative error of the standard deviation for Multiflow. Just as before for average delay, as flow size increases, the relative error decreases significantly; for flows greater than size 100, the median error in standard deviation is less than 27%. The graph is omitted due to lack of space.

V. CONCLUSION

Customers today are frustrated whenever their applications experience performance problems and demand better service from their ISPs. Network operators therefore need sophisticated tools to diagnose these problems whenever they do occur. In this paper, we have proposed a way to retrofit per-flow latency estimates in the NetFlow framework. We started with the fundamental observation that NetFlow already records two timestamps on a per-flow basis. By harnessing hash-based sampling framework, which has been standardized by IETF, we proposed a Consistent NetFlow architecture that ensures that different routers record the same set of flows. From the different flow records collected from different router interfaces at a flow collector, we have shown the design of opportunistic estimator that can utilize the background flow delay samples to estimate the per-flow average delay and standard deviation. Using different delay models and backbone traces, we have shown that our Multiflow estimator can achieve significantly accurate estimates (about 20% median error for flows of size greater than 100 packets) of per-flow latencies under several realistic scenarios compared to prior approaches.

VI. ACKNOWLEDGMENTS

This work was supported in part by NSF Award CNS 08316547 and a grant from Cisco Systems.

REFERENCES

- [1] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," IETF, RFC 1157, 1990.
- [2] Y. Zhao, Y. Chen, and D. Bindel, "Towards unbiased end-to-end network diagnosis," in *ACM SIGCOMM*, 2006.
- [3] N. Duffield, "Simple network performance tomography," in *ACM/USENIX IMC*, 2003.
- [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] 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.
- [6] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," in *IEEE/ACM Transactions on Networking*, 2000.
- [7] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot, "Analysis of measured single-hop delay from an operational backbone network," *IEEE JSAC*, vol. 21, no. 6, 2003.
- [8] "Cisco NetFlow," http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html.
- [9] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a Better NetFlow," in *ACM SIGCOMM*, 2004, pp. 245–256.
- [10] R. R. Kompella and C. Estan, "The power of slicing in internet flow measurement," in *ACM/USENIX IMC*, May 2005.
- [11] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems*, vol. 21, pp. 270–313, 2003.
- [12] L. Yuan, C.-N. Chuah, and P. Mohapatra, "ProgME: towards programmable network measurement," in *ACM SIGCOMM*, 2007.
- [13] N. M. Yan, Y. Chen, D. Bindel, H. Song, and R. H. Katz, "An Algebraic Approach to Practical and Scalable Overlay," in *ACM SIGCOMM*, 2004.
- [14] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Accurate and efficient SLA compliance monitoring," in *ACM SIGCOMM*, 2007.
- [15] J. Mahdavi, V. Paxson, A. Adams, and M. Mathis, "Creating a scalable architecture for internet measurement," in *Proc. of INET'98*, 1998.
- [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.
- [17] N. Duffield, A. Gerber, and M. Grossglauser, "Trajectory engine: A backend for trajectory sampling," in *IEEE Network Operations and Management Symposium (NOMS)*, 2002.
- [18] S. Machiraju and D. Veitch, "A measurement-friendly network (MFN) architecture," in *ACM SIGCOMM INM workshop*, 2006, pp. 53–58.
- [19] 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.
- [20] N. Duffield, B. Claise, D. Chiou, A. Greenberg, M. Grossglauser, and J. Rexford, "A framework for packet selection and reporting," RFC 5474, March 2009.
- [21] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, "Sampling and filtering techniques for ip packet selection," RFC 5475, March 2009.
- [22] N. Hohn and D. Veitch, "Inverting sampled traffic," in *ACM/USENIX IMC*, 2003.
- [23] J. F. Reynolds, "The covariance structure of queues and related processes—a survey of recent work," *Adv. Appl. Prob.*, 1975.
- [24] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM Transactions on Networking*, 1995.
- [25] C. Shannon, E. Aben, kc claffy, and D. E. Andersen, "CAIDA Anonymized 2008 Internet Traces Dataset (collection)," <http://imdc.datcat.org/collection/1-06BX-2=CAIDA+Anonymized+2008+Internet+Traces+Dataset>.
- [26] "Abilene-I data set," <http://pma.nlanr.net/Traces/Traces/long/ipls/1/IPLS-KSCY-20020814-092000-0.gz>.
- [27] "YAF: Yet Another Flowmeter," <http://tools.netsa.cert.org/yaf/>.
- [28] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, 1993.