Verifiable Network-Performance Measurements

Katerina Argyraki
EPFL, Switzerland

Petros Maniatis
Intel Labs Berkeley, USA

Ankit Singla[*]
UIUC, USA

1. SETUP

Informally, we want to design a measurement system such that (1) a monitor can accurately estimate the loss and delay performance of each network domain that participates in the system, (2) a network domain cannot bias the measurement process to its advantage without being detected, and (3) it needs a reasonable, tunable amount of resources to participate. We propose such a system, called Network Confessional.

1.1 Terminology

A *domain* is a contiguous network that falls under one administrative entity; in the current Internet, a domain would refer to an edge network or a single Autonomous System (AS).

A *path* is a sequence of nodes, where each node corresponds to a border router of a domain, and the first and last node belong to edge domains (Fig. 1).

With respect to a specific path, a node can be either an *input* node (the even-numbered nodes in Fig. 1) or an *output* node (the odd-numbered nodes in Fig. 1). Two consecutive nodes are *peering*, if they belong to adjacent domains (e.g., nodes 1 and 2, or 3 and 4 in Fig. 1). The link between two peering nodes i and j is *faulty*, if it introduces packet loss, or reordering, or delay beyond a value Δ_{ij} that is pre-negotiated between the two nodes (e.g., is characteristic of the link technology between them).

Each packet is associated with a specific path k, i.e., it is forwarded along path k until it reaches the last node on k or it is dropped. Given an input node i and an output node j that comes after i on path k, we denote by λ_{ij}^k the amount of packet loss experienced by path-k traffic between nodes i and j; we denote by $\delta_{ij}^k(q)$ the q-th quantile of the delay experienced by path-k traffic between i and j—for instance, if $\delta_{ij}^k(95) = 10$ msec, this means that 95% of the packets from path k that traverse i and j, experience delay below 10 msec between i and j.

1.2 Problem Statement

Each path k is associated with an entity called a *monitor*. Time is divided in fixed intervals, and at the end of each interval the path-k monitor and all the path-k nodes participate in a *measurement protocol*. After running the protocol, for each input node i and output node j that comes after i on path k, the path-k monitor computes an estimate of λ_{ij}^k, denoted by $\tilde{\lambda}_{ij}^k$, and an estimate of $\delta_{ij}^k(q)$, denoted by $\tilde{\delta}_{ij}^k(q)$.

We say that node i is *correct* with respect to (w.r.t.) path k, if it participates in the measurement protocol with the path-k monitor as specified. We say that node i is *detectably faulty* w.r.t. path k, if it breaks the measurement protocol in a way that affects a correct path-k node; the term "detectably faulty" was introduced and formally defined in the context of a distributed system in [11, 12]. When a sequence of nodes are detectably faulty w.r.t. path k, we call them *accomplices* w.r.t. path k. We say that node i is *exposed* to its peering node j w.r.t. path k, if: node j is correct, node i is detectably faulty and breaks the measurement protocol in a way that affects j, and node j detects i's behavior.

We want to design the measurement protocol, such that all of the following conditions hold:

1. If nodes i and j are correct w.r.t. path k, then
$$|\tilde{\lambda}_{ij}^k - \lambda_{ij}^k| < l_{ij}^k, \ \ |\tilde{\delta}_{ij}^k - \delta_{ij}^k| < d_{ij}^k \quad w.p. \quad \pi_{ij}^k$$
The error margins, l_{ij}^k and d_{ij}^k, and the probability with which they are honored, π_{ij}^k, depend on (and can be computed from) the rate of path-k traffic and node i's and node j's configuration—in particular the amount of memory and computing cycles used by the two nodes to run the protocol.

2. If node i is detectably faulty w.r.t. path k, then some accomplice of i is exposed w.r.t. path k to its peering node.

3. If node i is correct w.r.t. all paths it participates in, it does not need to maintain per-packet, per-flow, or per-path state to run the protocol.

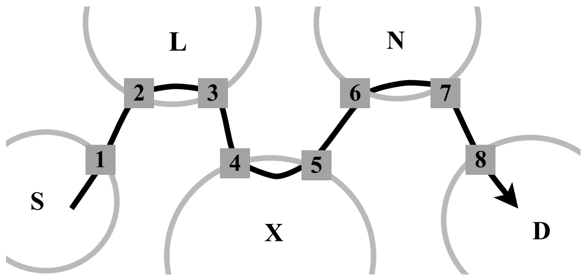

Figure 1: Circles represent administrative domains. The numbered boxes represent border routers.

[*]Ankit contributed to this work when he was a research assistant at EPFL.

All symbols mentioned above are summarized in Table 1. For brevity, when it is obvious from the context that we are referring to a particular path, we drop the superscript k from λ_{ij}^k and δ_{ij}^k.

Discussion. The first condition ensures that a monitor can estimate the performance of correct domains with probabilistic guarantees. The second condition ensures that, if a domain deviates from the protocol in a way that affects another domain (e.g., makes its performance appear worse than it is), then the misbehaving domain is exposed to the affected domain. For instance, suppose that node 5 in Fig. 1 tries to hide the fact that X is dropping packets by making it appear as if it delivers these packets to node 6; as long as node 6 is correct, according to the third condition, domain N will detect X's misbehavior. The third condition—no per-packet, per-flow or per-path state—is important, because a node may observe hundreds of thousands, perhaps even millions of concurrent flows and paths.

1.3 Assumptions

We make the following assumptions:

(1) There exists a way for each node to reliably exchange messages with a monitor, such that the authenticity and integrity of each received message is guaranteed. One way of realizing this assumption would be for each domain to establish an HTTPS connection with the monitor. It is possible to design more efficient dissemination mechanisms, but that is outside the scope of this paper.

(2) There exists a way for each of two peering nodes to debug the inter-domain link between them and determine whether it is faulty.

(3) Nodes (whether correct or faulty) do not apply any transformation to the observed traffic other than packet loss, delay, or reordering. In particular, they do not inject new packets or modify observed packets.

(4) Each node (border router or middlebox attached to a border router) can perform at wire speed simple per-packet operations. These include packet timestamp generation, arithmetic calculations or digest computations on a small, fixed portion of a packet, and modification of local state in a buffer.

2. WHY A NEW PROTOCOL

Instead of describing our protocol from scratch, we first build, in this section, "obvious" solutions by extending existing techniques, and explain why these do not meet the conditions of our problem statement. We close with a brief overview of Network Confessional.

Packet Obituaries+. As a first-cut solution, we consider the following modest extension to the Packet Obituaries protocol [3]. Each node computes a *receipt* for every observed packet, which consists of a *digest* for the corresponding packet and the *timestamp* for when the packet was observed. The path-k monitor collects all the receipts computed by all the path-k nodes; to estimate λ_{ij}^k, it counts how many path-k packets were observed at node i versus node j; to estimate δ_{ij}^k, it compares the timestamps recorded for the same packet at node i versus node j. Moreover, the monitor determines that peering nodes i and j are *inconsistent* w.r.t. a certain packet, if: (1) node i computed a receipt for the packet, and node j did not; (2) the difference in the two timestamps recorded for this packet at nodes i and j exceeds a value Δ_{ij} pre-negotiated between the two nodes. When the monitor detects such an inconsistency, it notifies both involved nodes (hence, if the inconsistency is the result of one node being detectably faulty, that node is exposed to its peering node).

This protocol fails to meet our third condition: it requires storing, processing, and disseminating per-packet receipts, leaving no room to a participating domain to choose (and tune, according to network conditions) the amount of resources it devotes to reporting its performance.

Coordinated Trajectory Sampling. Since the fundamental problem with Packet Obituaries+ is maintaining per-packet state, the first solution that comes to mind is to sample, i.e., produce receipts not on all packets, but on a representative subset, and use them to infer statistics for the rest. Hence, we first consider the following simple combination of Packet Obituaries and Trajectory Sampling [7] (POTS, for brevity).

Each node applies a uniform hash function to a small, fixed portion of each observed packet; if the outcome is equal to a pre-configured value, then the packet is sampled and a receipt is computed for it (note that, since all nodes use the same sampling function, they all sample the same packets). The path-k monitor collects all receipts computed by all path-k nodes; it estimates the loss and delay experienced by all packets, based on the loss and delay experienced by a representative subset of sampled packets [17]; it determines and reports inconsistencies on sampled packets as in Packet Obituaries+.

This protocol fails to meet our first condition: it is possible that all nodes are correct, yet a monitor's estimates are arbitrarily inaccurate. In particular, each input node can engage in the following behavior: for each observed packet, determine whether the packet should be sampled and, if yes, treat the packet preferentially, e.g., assign it to a high-priority queue. I.e., the nodes bias the sampling process, such that they tell the truth about what happens to the sampled packets, but that is not representative of what happens to the rest of the traffic.

Note that, even if we modify POTS, such that domains sample non-overlapping subsets of packets, domains can collude, such that all of them treat all subsets of sampled packets preferentially. In this way, all nodes are correct, yet a monitor's estimates can still be arbitrarily inaccurate.

Verifiable Aggregation. An alternative to sampling is aggregation: instead of computing receipts for sampled packets, compute receipts for packet aggregates. Hence, we next consider a combination of Packet Obituaries and the Lossy Difference Aggregator (LDA) [13]. Note that we could have equally used the "Secure Sketch" technique from [10]—the

Symbol	Meaning
λ^k_{ij}	Packet loss experienced by path-k traffic between nodes i and j.
$\delta^k_{ij}(q)$	q-th quantile of delay experienced by path-k traffic between nodes i and j.
l_{ij}	Maximum estimation error incurred by path-k monitor when estimating λ^k_{ij}.
d_{ij}	Maximum estimation error incurred by path-k monitor when estimating $\delta^k_{ij}(q)$.
π_{ij}	Probability with which l_{ij} and d_{ij} are honored.

Table 1: Defined symbols.

conclusion would have been the same.

Each node divides the observed path-k traffic into "segments" (finite-length sequences of consecutively observed packets), then divides packets from each segment into aggregates based on their content (e.g., by applying a uniform hash function to a small portion of each packet). It produces a receipt for each aggregate in each segment, which consists of an aggregate identifier, a packet count, and an average timestamp. For simplicity, assume that all nodes produce receipts on non-overlapping aggregates.

The path-k monitor collects all the receipts computed by all path-k nodes; to estimate λ^k_{ij}, it counts the number of packets observed at node i versus node j; to estimate δ^k_{ij}, it relies on the LDA technique (LDA does not estimate delay quantiles, only average delay; however, one can imagine an LDA extension, which would approximate the delay distribution between the two nodes by combining average-delay information from multiple aggregates). Moreover, the monitor determines that peering nodes i and j are inconsistent with respect to a segment, if: node i reported observing more packets from that segment than node j, or the difference in the two average timestamps recorded for this segment at nodes i and j exceeds a value Δ_{ij} pre-negotiated between the two nodes. When the monitor detects such an inconsistency, it notifies both involved nodes.

This protocol fails to meet our third condition: it requires maintaining per-aggregate state (and recall that there exist multiple aggregates per path). This is a fundamental limitation of aggregation-based solutions—to produce receipts on aggregates, nodes have to collect per-aggregate statistics, which requires maintaining at least one record per active aggregate, on fast memory that can be updated at line speed.

Our Solution. We employ sampling, but in a way that is not susceptible to bias. Our solution shares elements with Trajectory Sampling (nodes produce receipts for a subset of observed packets and choose which packets to sample using hash functions), but prevents sampling bias in the following way: the sampling function is keyed on *future* traffic, making the samples unpredictable. Specifically, a domain does not know whether it will have to report measurements on a particular packet until after it has forwarded that packet to its downstream neighbor. As a result, an unscrupulous domain has no way to decide whether to "sugarcoat" its performance by preferentially treating particular packets. The challenge is implementing this idea in a practical manner, i.e., without requiring the source to explicitly signal to all the other nodes which packets to sample, in accordance to the memory and computing requirements dictated by our third condition, and with per-domain tunability.

3. BASIC OPERATION

We now describe the basic elements of Network Confessional. For simplicity, we assume, in this section, that all nodes have synchronized clocks and that there is no ambiguity regarding the path followed by a packet (i.e., when a node observes a packet, it knows which path this packet is associated with). We relax these assumptions in the next section.

3.1 Node Operation

Each node samples a subset of the packets it observes and generates a receipt for each sampled packet. A receipt has form $\mathcal{R} = \langle ReporterID, ReporterConfig, PacketID, Time, NeighborID, \Delta \rangle$. $ReporterID$ is the identity of the reporting node and $ReporterConfig$ a specification of its sampling function (more on this later). $PacketID$ is a digest of the packet's headers and a small portion of its content. $Time$ specifies when the packet was observed. $NeighborID$ is the identity of the node that is peering with the reporter on the path where the packet belongs. Δ is a value agreed upon between the reporter and the neighbor; it is meant to lower-bound the difference in timestamps one should expect between the two nodes.

Instead of sampling packets in real time, each node collects state on *all* observed packets, but only for a fixed, *short* period of time (milliseconds or so). The node is periodically told which of the stored per-packet state to keep and which to discard. Since a domain learns whether a packet's fate will affect estimates of its performance only *after* it has forwarded that packet, it cannot treat sampled packets preferentially.

A key question is *who* tells each node which packets to sample. One approach would be to use explicit signaling; for example, in Fig. 1, domain S could explicitly tell all nodes which packets to sample from the packet stream sent from S to D, as in the PAAI-1 packet-dropping adversary identification protocol [18]. In our context, however, that approach would be naïve, because it would require each source domain to actively probe all Internet paths through which it sends traffic; it would also require each node to implement fine-granularity per-packet timers. Instead, each node decides whether to sample a packet based on the *contents of another packet* observed *later*. In this sense, domain S implicitly dictates which of its packets should be sampled, through the traffic it sends out subsequently.

More specifically, each node maintains a circular buffer, where it stores a tuple (path ID, packet ID, and timestamp) for the β most recently observed packets. Alg. 1 shows what happens when a node observes a new packet p. First, the node computes a tuple T_p for the new packet (line 1). Then,

Algorithm 1 *ProcessPacket*(packet p)

$PathID$(packet)	packet's path
$PacketID$(packet)	hash function
$MarkerID$(packet)	hash function
$Hash$(packet$_1$, packet$_2$)	hash function
μ	marking threshold
σ	sampling threshold
$Buffer$	circular buffer
Initially	$Buffer \leftarrow \emptyset$

```
1:  $T_p \leftarrow \langle PathID(p), PacketID(p), Time \rangle$
2:  **if** $MarkerID(p) < \mu$ **then**
3:      **for all** $T$ in $Buffer$ **do**
4:          **if** $T.PathID = T_p.PathID$ **then**
5:              **if** $Hash(T.PacketID, T_p.PacketID) < \sigma$ **then**
6:                  Copy $T$ for dissemination
7:              Remove $T$ from $Buffer$
8:      Copy $T_p$ for dissemination
9:  **else**
10:     Add $T_p$ to $Buffer$
```

if the packet satisfies a certain condition, it is chosen as a "marker" packet (line 2). In that case, its contents determine which of the β most recently observed packets to sample (lines 3–5); only packets from the same path with the marker packet can be sampled (line 4). The tuples of the chosen packets are copied for later dissemination (line 6). All tuples that correspond to packets from the same path with the marker are removed from the buffer (line 7). The marker packet itself is also sampled (line 8). If the new packet is not chosen as a marker, its tuple is added to the circular buffer (lines 9, 10).

The parameters of the algorithm are: the size of the circular buffer β, the *marking threshold* μ, which determines which packets are markers (line 2), and the *sampling threshold* σ, which determines which of the tuples in the circular buffer to sample (line 5). Moreover, $MarkerID(p)$ is a function that provides uniform hashing between 0 and some maximum value $\mathcal{M}$, while $Hash\ (PacketID(p_1),\ PacketID(p_2))$ provides uniform hashing between 0 and some maximum value $\mathcal{S}$.

Lemma 3.1. *Consider a path that forms a fraction π_p of the total traffic observed by a node. If the node uses Alg. 1, it samples each packet from that path with probability*

$$\pi_s = \left(1 - \left(1 - \pi_p \frac{\mu}{\mathcal{M}}\right)^{\beta}\right) \cdot \frac{\sigma}{\mathcal{S}} \qquad (1)$$

PROOF. An observed packet p is sampled when: (1) p's tuple is still in the circular buffer when the next marker m from the same path arrives and (2) $Hash\ (PacketID(p), PacketID(m)) < \sigma$. We first compute the probability of event (1). Consider an observed packet p that is *not* chosen as a marker. Each of the packets observed after p is from the same path with p *and* is chosen as a marker with probability $\pi_p \frac{\mu}{\mathcal{M}}$. Hence, the number of packets observed between p and m (we call it the "distance" between p and m) is a random variable with geometric distribution and success rate $\pi_p \frac{\mu}{\mathcal{M}}$. It follows that the probability that $Distance(p, m) < \beta$ is equal to the cumulative distribution function (CDF) of the geometric distribution, i.e., $1 - \left(1 - \pi_p \frac{\mu}{\mathcal{M}}\right)^{\beta}$. Next, we compute the probability of event (2) given event (1). Given that p' tuple is still in the circular buffer when m arrives, p is sampled with probability $\frac{\sigma}{\mathcal{S}}$. Hence, packet p is sampled with probability $\left(1 - \left(1 - \pi_p \frac{\mu}{\mathcal{M}}\right)^{\beta}\right) \cdot \frac{\sigma}{\mathcal{S}}$. $\square$

Eq. 1 says that, as long as $\beta \gg \frac{\mathcal{M}}{\mu} \cdot \frac{1}{\pi_p}$, then $\pi_s \approx \frac{\sigma}{\mathcal{S}}$, i.e., a node samples each observed packet p with the same probability $\frac{\sigma}{\mathcal{S}}$, independently from which path p is associated with, and independently from the size of the circular buffer β. Intuitively, as long as the circular buffer is large enough that a packet p's tuple is always in the buffer when the next marker from the same path with p is observed, then the size of the circular buffer β does not affect which packets are sampled. For the rest of this section, we will assume that this is the case. In Section 4.1, we will see how to choose the various parameters of the system to make this hold.

The marking threshold μ is a system-wide constant, common for all nodes; hence, all nodes on a certain path select the same packets as marker packets for that path (modulo loss). In contrast, the sampling threshold σ is a local parameter, chosen independently at each node. If all nodes on a certain path choose the same σ, they all sample the same packets from that path (modulo loss and reordering). We turn next to what happens when different nodes select different σ.

3.2 Tunability

Each node chooses its own sampling threshold σ. At the same time, given any number of nodes and their sampling thresholds, we maximize the number of packets that are commonly sampled by all nodes on the same path. The key element that enables this property is the inequality in line 5 of Alg. 1. Consider nodes 1 and 2, with different sampling thresholds σ_1 and $\sigma_2 > \sigma_1$. Suppose there is no packet loss or reordering between the two nodes; p is a packet sampled by node 1, and m is the first marker from the same path with p, observed after p. Since node 1 samples p, this necessarily means that $Hash(PacketID(p), PacketID(m)) < \sigma_1 < \sigma_2$, which means that node 2 also samples p.

So, even though each node chooses its sampling threshold σ independently, if there is no packet loss or reordering between two nodes on the same path, the node with bigger σ will sample at least all the packets from that path sampled by the node with smaller σ.

3.3 Monitor Operation and Statistics

At the end of each time interval, the path-k monitor collects all the receipts produced by path-k nodes. We now consider the monitor associated with the path in Fig. 1 and describe how it estimates and verifies the performance of domain X over a given time interval.

Loss Estimation. For brevity, we define $\lambda = \lambda_{45}$. For simplicity, we first assume that there is no packet reordering within domain X, i.e., packets that are not lost between nodes 4 and 5 are observed at the two nodes in the same order. We will remove this assumption later.

The monitor considers the receipts R_4 and R_5 generated by the two nodes during the given time interval. By looking at the $\mathit{ReporterConfig}$ values of these receipts, it divides the time interval into sub-intervals, such that, throughout each sub-interval, each node used a constant sampling threshold. For each sub-interval, it counts the number of packets K that were sampled by node 4 and should have been sampled by both nodes, i.e., all packets p that: (1) were sampled by node 4 based on a marker m that was also observed at node 5 and (2) satisfy $\mathit{Hash}(\mathit{PacketID}(p),\ \mathit{PacketID}(m)) < \sigma_{min}$, where σ_{min} is the smaller sampling threshold used by the two nodes. Of these K packets, it counts the number of packets k that were *not* sampled by node 5 and estimates the loss rate λ between the two nodes as $\tilde{\lambda} = \frac{k}{K}$.

Now assume that there *is* some packet reordering between the two nodes. As above, the receipt collector first counts the number of packets K that were sampled by node 4 and should have been sampled by both nodes and, of these, the number of packets k that were *not* sampled by node 5. Of these k packets, let's say that k_l were lost between nodes 4 and 5, while $k_r = k - k_l$ were reordered with their previous or next marker such that node 5 did observe them but did not sample them. Hence, to accurately estimate the loss rate between the two nodes (as $\frac{k_l}{K}$), the receipt collector would need to know k_l or k_r.

Fortunately, there is a simple way around this problem. Packet reordering caused node 5 not to sample k_r packets that it would have sampled otherwise, but it also caused node 5 to *sample* $\bar{k_r}$ packets that it would *not* have sampled, had there been no reordering. Assuming that the probability of two packets being reordered depends only on the distance between them [8], then k_r and $\bar{k_r}$ should be statistically the same. The receipt collector does not know k_r (it is masked by the k_l packets that were lost between the two nodes), but it does know $\bar{k_r}$; it is the number of packets p that: (1) were not sampled by node 4, (2) were sampled by node 5 based on a marker m that was also observed at node 4, and (3) satisfy $\mathit{Hash}(\mathit{PacketID}(p), \mathit{PacketID}(m)) < \sigma_{min}$. Hence, the receipt collector computes $\bar{k_r}$ and estimates the loss rate λ between the two nodes as $\tilde{\lambda} = \frac{k-\bar{k_r}}{K}$, i.e., it approximates k_r with $\bar{k_r}$.

Lemma 3.2. *The expected value of the estimate is λ. The relative standard deviation is*

$$\sqrt{\frac{1-\lambda}{N\ \pi_s\ \lambda} + \frac{2\ \pi_r(1-\pi_r)}{N\ \pi_s\ \lambda^2}} \qquad (2)$$

where all the parameters are specified in Table 2.

Parameter	Meaning
λ	Actual loss rate (that we are trying to estimate).
N	Number of packets observed at node 4 during the given sub-period before a marker m that was also observed at node 5.
π_s	Probability that a packet is sampled, given by Eq. 1.
π_r	Probability that a packet is reordered with its marker and observed at node 5 but not sampled by it.

Table 2: Parameters for Lemma 3.2.

PROOF. In the appendix. □

Once we know the standard deviation of the estimate, it is straightforward to compute its maximum distance from the actual loss with a given probability π [16].

Lemma 3.2 tells us that packet reordering does not prevent us from estimating λ correctly, however, it does increase the relative standard deviation of our estimate. The relative standard deviation depends on the average number of sampled packets that we use to produce the estimate ($N\ \pi_s$ in Eq. 2): the better (lower) the relative standard deviation that we want to achieve, the more samples (receipts on sampled packets) we need to collect. To give some concrete numbers, suppose that $\lambda = 5\%$, and we want to estimate it with a relative standard deviation of 0.1. According to Eq. 2, if there is no packet reordering ($\pi_r = 0$), we can produce a new estimate every time we have collected receipts on $N\ \pi_s = 1900$ new packets; if there *is* packet reordering, such that $\pi_r = 10\%$ of the packets that should be sampled by node 5 miss their marker and are not sampled, then we can produce an estimate every time we have collected receipts on $N\ \pi_s = 9100$ new packets. Assuming a traffic rate of 100 Mbps, a sampling rate of $\pi_s = 1\%$, and about 400 bytes/packet, 1900 sampled packets correspond to 7 seconds, while 9100 packets correspond to 30 seconds. So, packet reordering forces us to estimate loss rate at longer intervals in order to achieve a given level of accuracy.

Delay Estimation. The receipt collector considers all the receipts generated by nodes 4 and 5 during a given time period. By looking at the $\mathit{PacketID}$ of these receipts, it determines the set of packets that were commonly sampled by the two nodes. By comparing the Time reported by the two nodes for each commonly sampled packet, it computes the delay incurred by the packet within X. Finally, by combining the delay incurred by multiple packets, it estimates the maximum delay incurred by $q\%$ of the packets, by using the algorithm proposed in [17]. That algorithm takes as input (1) the delays incurred by all sampled packets, (2) the quantile q we are interested in, and (3) a probability π, and outputs a lower and upper bound, such that the actual delay value we are estimating falls between the two bounds with probability π.

Verification. The monitor considers all the receipts generated by each pair of peering nodes i and j during the given time interval. Then it identifies the set of packets that were sampled by node i and should have been sampled by both nodes (we explained how this is achieved in the "Loss Estimation" paragraph above). The monitor determines that nodes i and j are *consistent* with respect to a packet p, if: (1) Either both or none of them provide a receipt on p. (2) If both nodes provide receipts on p (say, $R_i(p)$ and $R_j(p)$), then:

$$\begin{aligned} R_i(p).\Delta &= R_j(p).\Delta \\ R_i(p).Time - R_j(p).Time &\leq R_j(p).\Delta \end{aligned}$$

These rules express the fact that a correct inter-domain link does not introduce loss or unpredictable delay. If two peering nodes are not consistent with respect to any packet, the monitor reports the inconsistency to both of them.

This means that, as long as peering nodes i and j use the same sampling threshold (hence, are expected to sample the same packets), if node i is not correct (i.e., tries to blame its problems on j), while node j is correct, that will necessarily lead to an inconsistency and expose i to j. However, if node i uses a larger sampling threshold (hence, is expected to sample more packets) then node i is free to lie about the packets that should be sampled by i but not by j. In short, the monitor can verify X's performance, only based on the packets that are expected to be commonly sampled by X and its neighbors.

4. ANALYSIS

Network Confessional tries to (1) prevent nodes from biasing their sampling and (2) maximize the number of packets commonly sampled by all nodes. We now look at how it reacts when node behavior undermines these two goals.

4.1 Delayed Forwarding

Consider an input node on path k, which tries to cheat in the following way: it briefly stores each path-k packet before forwarding it, in case the next path-k marker arrives soon enough for the node to determine whether the packet needs be sampled and treat it accordingly. In this way, the node manages to treat a fraction of the sampled path-k packets preferentially, hence exaggerate its domain's performance w.r.t. path k.

To prevent domains from exaggerating their performance in this way, the path-k monitor can discard any receipt produced by a node i, if the receipt corresponds to a packet observed within a time interval τ before the next marker (at node i). E.g., by choosing $\tau = 100$ msec, the monitor ensures that, to learn that a packet needs to be sampled and treat it preferentially and get away with it, a node would have to first delay that packet by at least 100 msec.

The question then is, what fraction of the collected path-k receipts does the monitor get to keep? The answer is equal to the probability that a path-k packet is *not* observed within an interval τ before the next path-k marker:

$$\phi = \left(1 - \frac{\mu}{\mathcal{M}}\right)^{rate \times \tau} \tag{3}$$

where $rate$ is the packet rate of path k. Intuitively, the higher the frequency of markers, $\frac{\mu}{\mathcal{M}}$, and the higher the packet rate of path k, the more path-k packets are observed time-wise close to the next path-k marker, hence the more the receipts that the path-k monitor has to discard.

We want the fraction ϕ of valid (non-discarded) receipts to be non-negligible, but we cannot achieve that for all packet rates: for a given minimum fraction ϕ and a given frequency of markers, $\frac{\mu}{\mathcal{M}}$, there exists a maximum packet rate that path k must honor in order for the path-k monitor to collect a non-negligible number of receipts and compute accurate statistics for the path. Differently said, the value of the marking threshold μ determines the maximum packet rate for which the monitors can compute accurate statistics. For example, assuming $\tau = 100$ msec and $\phi = 10\%$, if $\frac{\mu}{\mathcal{M}} = 0.00009$, then our system supports a maximum per-path packet rate of 250 000 packets/sec (about 1 Gbps, assuming 500-byte packets).

At the same time, as argued in Section 3.1, path k must contribute a minimum fraction of the packets observed at each node i in order for the path-k monitor to compute accurate statistics for the path: from Eq. 1, if the probability of a packet observed at node i belonging to path k is π_p, node i must use $\beta \gg \frac{\mathcal{M}}{\mu} \cdot \frac{1}{\pi_p}$ in order for our analysis of Section 3.3 to hold. Differently said, given a marking threshold μ, the circular buffer size β used by node i determines the minimum π_p for which the monitors can compute accurate statistics. For example, given $\frac{\mu}{\mathcal{M}} = 0.00009$, if node i uses $\beta = 5\,000\,000$ tuples, then our system supports a minimum $\pi_p = 0.02$ for node i.

To summarize, the value of the marking threshold μ and the circular buffer sizes used by the path-k nodes determine the minimum and maximum packet rate that path k must honor so that the corresponding monitor can compute accurate statistics for the path. The next question then is, what are reasonable values for today's networks? Table 3 shows the range of per-path rates that our system supports for different marker frequencies $\frac{\mu}{\mathcal{M}}$, assuming a circular buffer that can fit $\beta = 5$ million tuples. The minimum π_p is computed such that $\beta \gg \frac{\mathcal{M}}{\mu} \cdot \frac{1}{\pi_p}$. The minimum rate is computed from π_p, assuming a saturated OC-192 network interface, i.e., a bit rate of 10 Gbps or a packet rate of 2.5 million packets per second.[1] The maximum rate in packets per second is computed from Eq. 3, for $\tau = 100$ msec and $\phi = 10\%$; it is converted to Mbps assuming 500-byte packets.

Table 3 shows that we are facing a resource challenge. We show numbers for $\beta = 5$ million tuples, because that corresponds to a few MB of memory, which can fit within a single SRAM chip, making our system easier to implement. Ideally, we would want to use one such small buffer per node, even for nodes that correspond to OC-192 interfaces. However, doing so, would not allow us to cover a large enough range of per-path packet rates—we could cover only one of the four ranges stated in the table, which would

[1]We assume a saturated OC-192 interface in order to be conservative. If we assume a lower-rate and/or under-utilized interface, we cover a larger range for the same parameter values.

$\tau = 100$ msec, $\phi = 10\%$, $\beta = 5$ million tuples			
$\frac{\mu}{\mathcal{M}}$	Min π_p	Min rate, OC-192 pps (Mbps)	Max rate pps (Mbps)
0.00009	0.02	50 000 (200)	250 000 (1000)
0.00045	0.004	10 000 (40)	50 000 (200)
0.0022	0.0008	2000 (8)	10 000 (40)
0.01	0.00018	450 (1.8)	2000 (8)

Table 3: Range of supported per-path rates.

not be enough. On the other hand, to cover the four ranges with a single buffer, we would need $\beta = 500$ million tuples ($\beta \gg \frac{1}{0.00009} \cdot \frac{1}{0.00018}$), i.e., hundreds of memory chips, which would make our system practically infeasible.

The solution to this resource challenge is to use multiple buffers, each with different parameters, for each node. In particular, each node has N circular buffers, each operating with a different marking threshold; when a new packet is observed at the node, the node runs Alg. 1 *for each different circular buffer*, i.e., each observed packet causes the node to change the state of *all* the circular buffers. However, in the end, each buffer yields accurate statistics for a different set of paths. For example, to cover all the four ranges of Table 3, we need to use $N = 4$ circular buffers per OC-192 interface, each buffer corresponding to one row of the table and covering the corresponding per-path packet-rate range.

4.2 Other Adversarial Behavior

Deliberate Marker Loss. An under-performing domain (say X in Fig. 1) may drop all marker packets, causing the next domain (N, in our example) to not sample any packets, in order to ensure that X's performance is never verified according to N's receipts or simply to make N look bad. Such behavior is necessarily exposed, *because all marker packets must be sampled*. If X drops a marker m, it either has to admit dropping it, or lie and be inconsistent with N's report that it never received m. So, if X consistently drops marker packets, it either admits it and is globally exposed as under-performing and misbehaving, or blames the losses on N and is exposed to N as a liar.

Packet Crafting. Network Confessional was not designed to resist attacks where domains deliberately inject or modify packets (Section 1), however, we do discuss what happens when it encounters such behavior. Suppose domain X modifies an observed packet p or, equivalently, drops p and introduces a new packet q in its place. Let's first assume that q has a different packet ID from p. There are several cases: (1) p is sampled, in which case, node 4 produces a receipt on p, but node 5 does not, hence X appears to have dropped p. (2) q is sampled, in which case, X appears to have introduced a new packet. (3) Neither packet is sampled, in which case, X's behavior does not impact its receipts. So, it *is* possible for X to modify a few packets and get away with it. However, if it consistently engages in such behavior, given that it cannot predict which packets will be sampled, it *will* eventually have to report on one of the modified packets, and its behavior will be exposed. On the other hand, if X can craft q such that it has the same packet ID with p, then its behavior does not impact its receipts at all. We believe that we can defend against such attacks by making it hard to craft packets with a given packet ID, but we defer this to future work.

Now suppose domain X modifies the TTL field of the IP header of an observed packet p, such that p is dropped within domain N; as a result, N appears (based on its receipts) to be under-performing. Network Confessional cannot detect such malicious behavior, and it is not meant to; it is meant to detect where packets are dropped or delayed, and this is precisely what it does in this case, even if the reason for the drop is not really N's fault. Of course, if X consistently engages in such behavior, N will eventually detect that it is dropping packets, investigate the problem, and trace it back to X delivering packets with low TTL. In this sense, Network Confessional does exactly what it is meant to—detect packet loss and unpredictable delay and alert the involved domains; how each domain responds and investigates depends on its policy.

5. PRACTICAL CONSIDERATIONS

We now relax the assumptions made in Section 3—that all nodes must have perfectly synchronized clocks and that there is no ambiguity regarding the path followed by each observed packet.

Clock Synchronization. Network Confessional does not dictate any particular clock-synchronization policy. However, it is to each participating domain's best interest to keep its reporting nodes (its border routers) reasonably synchronized, since the domain's delay performance will be estimated based on the timestamps reported by these nodes. Moreover, it is to two neighboring domains' best interest to keep peering nodes reasonably synchronized, otherwise their timestamp difference will exceed the reported Δ, and the two neighbors will generate inconsistent reports (hence appear to have a problematic inter-domain link or be involved in a lie). We should clarify that domains are free to report arbitrarily large Δ values: nothing prevents nodes 3 and 4 from reporting a Δ of seconds between them, hence not needing to synchronize their clocks beyond that granularity. However, that does make it look like they are connected through an awfully slow inter-domain link—not a good feature to advertise to their customers and peers.

So, what is a reasonable granularity at which a domain should keep its border routers synchronized? Since typical intra-domain latency is on the order of tens of milliseconds, a granularity of a few milliseconds is sufficient. This is reportedly achievable with NTP [5]. But if NTP is not deemed sufficiently reliable, a domain can equip its border routers with radio or GPS receivers [1], currently costing \$200 a piece—a negligible cost compared to that of a border router.

Mapping Packets to Paths. In reality, a node cannot know the path followed by each observed packet, so it clas-

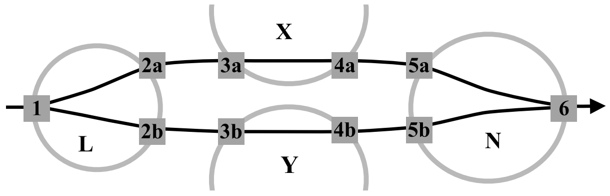

Figure 2: Domain S load-balances traffic with the same source and destination prefix across two inter-domain paths.

sifies packets per {source prefix, destination prefix} pair, where "prefix" is the *origin* prefix of the corresponding IP address as obtained through BGP, such that all domains that observe a packet p derive the same prefix pair for the packet. This has no implication for us when all packets with the same source and destination prefix follow the same path; otherwise, it requires a straightforward extension that we describe next. We should clarify that this extension is not needed by domains that apply the common types of load-balancing. For instance, domains can load-balance traffic per destination prefix or source/destination prefix across multiple inter-domain paths without the extension. It is only needed by domains that load-balance traffic *with the same source and destination prefix* across different inter-domain paths; we are not aware of ISPs engaging in such load-balancing, but there is no way to verify that they do not.

First, we explain why load-balancing traffic with the same $PathID$ across multiple inter-domain paths is problematic for Network Confessional. Consider the scenario depicted in Figure 2. Suppose node 1 observes packet sequence $\langle p_1, p_2, p_3, p_4, m_1, p_5, p_6, p_7, p_8, m_2\rangle$ and samples packets p_1, p_4 based on marker m_1 and packets p_5, p_8 based on marker m_2. Now suppose that node 1 load-balances the observed sequence across nodes 2a and 2b, such that 2a observes packet sequence $\langle p_1, p_3, m_1, p_6, p_8\rangle$ and samples packet p_1 based on marker m_1, while 2b observes packet sequence $\langle p_2, p_4, p_5, p_7, m_2\rangle$ and samples packets p_2, p_5 based on marker m_2. So, load-balancing has a similar effect to packet reordering: First, some packets are sampled at L's entry point and successfully delivered to the next domain(s), yet not sampled by any of its exit points (packets p_4 and p_8); we say that these packets were *downgraded*. Second, some packets are *not* sampled at L's entry point, yet *are* sampled at one of its exit points (packet p_2); we way that these packets were *upgraded*. As a result, a receipt collector cannot just count how many packets were sampled at L's entry point but not at its exit points as lost, because that would over-estimate L's loss.

We now describe how a receipt collector estimates L's loss during a certain time period. In a nutshell, it performs the same trick that we used to deal with packet reordering (Section 3.3), i.e., leverages the fact that the number of downgraded and upgraded packets should be statistically the same in order to estimate the number of downgraded packets and subtract it from the loss estimate. More specifically: Without loss of generality, we assume that during the given time period, nodes 1, 2a, and 2b use the same buffer size β and the same sampling threshold σ. The receipt collector performs the following operations: (1) It counts the number of packets K that were sampled at L's entry and should have been sampled at L's exit (all packets p that were sampled by node 1 based on a marker m that was also observed at either node 2a or node 2b). (2) Of these K packets, it counts the number of packets k that were not sampled at L's exit. These are the packets that were either lost in L or were "downgraded" due to load-balancing. (3) It counts the number of packets $\bar{k}$ that were sampled at L's exit but not at L's entry. These are the packets that were "upgraded" due to load-balancing. (4) It estimates the loss rate λ as $\lambda^* = \frac{k-\bar{k}}{K}$. A similar approach is used to estimate N's loss rate.

The accuracy of this estimate is exactly the one given by Lemma 3.2, with the difference that π_r is now the probability that a packet misses its marker due to either packet reordering or load-balancing.

Privacy. We will now argue informally that external observers can see no more internal information about a domain with Network Confessional than they can see today without it. We acknowledge that privacy deserves a fuller, formal analysis, but defer that to future work.

First, we consider the "privacy perimeter" lying around a single participating domain, i.e., consider whether Network Confessional exposes to the outside world any information that was previously exclusively known to the domain. Our privacy argument is based on the content of traffic receipts. The two receipt fields directly dependent on traffic are $PacketID$ and $Time$, both of which can be filled in by the domain's neighbor transmitting packets to or receiving packets from the domain. $ReporterID$, $NeighborID$, and Δ are already known to the corresponding neighbors, hence leak no information that was previously exclusively known to the domain.

Next, we consider the privacy perimeter lying around a pair of neighboring domains. As described in Section 3.1, traffic receipts do reveal some information that would otherwise remain private between the two neighbors: the number of peering points (as exposed via distinct $ReporterID$'s and $NeighborID$'s), as well as the expected delay imposed by the inter-domain links (as exposed via Δ). However, in practice, two neighbors can easily conceal both types of information from outsiders. First, they can conceal the number of peering points by using a single pair of $ReporterID$ and $NeighborID$ in traffic receipts. They can also conceal the actual delay of links in a similar fashion, as follows. They can agree to "absorb" the latency of the inter-domain link into their own intra-domain latencies. For instance, consider nodes 5 and 6 from Fig. 1 and assume the latency of the link between them is 1 msec. Instead of reporting $\Delta = 1$ msec, the two nodes report $\Delta \approx 0$. When node 5 observes packet p at time t_1, it reports observing it at time $t_1 + 0.5$ msec; similarly, when node 6 observes packet p at time t_2, it reports observing it at time $t_2 - 0.5$ msec. In this way, the latency of the inter-domain link is hidden from the outside world and "charged" equally to the two domains. Note that this does not affect in any way the capability of Network Confessional to detect and expose lies.

Partial Deployment. Partial deployment is still beneficial to the participating domains. Even if X is the only domain on a certain path that has deployed Network Confessional, its performance reports may not be verified by its neighbors, but they are still verifi*able*. So, during a congestion incident, X can still position itself as the "good" ISP that provides troubleshooting information to its customers—it is not its fault that the other ISPs on the path are not up to the task. X can even use this as an incentive to encourage multi-network customers to connect all their networks through X—since that way they avoid domains that do not provide troubleshooting information.

Incentives. If domain X has not deployed Network Confessional, but its neighbors have, then X's neighbors are free to blame their performance problems on X (since X does not produce any receipts to refute their claims). Consequently, the fault localization properties of Network Confessional are provided only at the granularity of deployment—informally, the sub-graph of the domain topology whose vertices are participating domains and whose edges link participating domains only over domain paths that include no participating domain in the topology. On the other hand, the loss of fault-localization resolution due to partial deployment can be viewed as an incentive for adoption: a domain has to report on its performance in order to prevent its neighbors from blaming their problems on it undetected.

6. IMPLEMENTATION

Hardware Implementation. We now outline one possible implementation of Network Confessional that requires an SRAM buffer and a small TCAM (ternary content addressable memory) chip per linecard. TCAM is already widely used in routers for storing forwarding and filtering tables, in general, any state that needs to be accessed at line rate. It is appropriate for such applications, because it can access *in parallel* all the entries of a stored table and return any matches within a few nanoseconds, independently of the table's size.

Each node uses two circular buffers per linecard: one for marker packets, stored in TCAM, and one for non-marker packets, stored in SRAM. Upon receiving a packet p, the node determines whether p is a marker packet, creates for it a $\langle PathID,\ PacketID,\ Time\rangle$ tuple, and adds it to the head of the marker or non-marker buffer, accordingly. In parallel, a separate process reads tuples from the tail of the non-marker buffer and determines whether each tuple T_p should be kept or discarded, as follows. First, it reads $T_p.PathID$ and $T_p.Time$, and identifies the corresponding marker tuple: it uses the TCAM search capabilities to identify all tuples T' in the marker buffer with $T'.PathID = T_p.PathID$, deletes any tuple with $T'.Time < T_p.Time$ from the marker buffer, and chooses the marker tuple T with the smallest timestamp that also satisfies $T.Time > T_p.Time$. Then, if $Hash$ $(T.PacketID,\ T_p.PacketID) < \sigma$, it copies T_p elsewhere for later dissemination, otherwise, it discards T_p.

The feasibility and cost of this implementation are determined by the sizes of the two buffers. The non-marker buffer must have size β chosen based on the analysis of Section 4.1. The marker buffer must have size significantly larger than $\frac{\mu}{\mathcal{M}}\beta$, so that the probability of an overflow (i.e., that a marker tuple is deleted before it is used) is negligible. For instance, for $\frac{\mu}{\mathcal{M}} = 0.01$, we need a non-marker buffer that can store around 1 million tuples and a marker buffer that can store around 0.1 million tuples.

Receipt Overhead. Each router that supports Network Confessional must periodically extract the sampled state from its data-path and export it in the form of receipts, akin to how a NetFlow-enabled router periodically extracts NetFlow records from its data-path and exports them to a management server for processing. The amount of memory, processing, and bandwidth required for this operation is directly proportional to the rate at which the router produces receipts, i.e., its sampling rate. This can be locally tuned to match the router's resources by changing the sampling threshold σ.

We have said that each domain makes each receipt available to every other domain that observed the corresponding traffic. Whether this happens pro-actively (through a constant receipt stream) or on-demand (e.g., through a secure web interface), receipt dissemination introduces, in each path, bandwidth overhead that depends on (1) the number of border routers on that path and (2) the rate at which each of these routers produces receipts. This may seem, at first, to be cause for concern—one could argue that introducing bandwidth overhead that grows with the total number of border routers per path is not a scalable approach. In practice, this is not a problem: Paths consist on average of 3–4 domains (hence 4–6 border routers), while most paths consist of fewer than 6 domains (10 border routers) [2]. Consider a 6-domain path, where each border router samples 1% of the path's packets. Assuming 20 bytes per receipt, this path will incur an overhead of 2 bytes per packet; assuming 400 bytes per packet, this leads to a 0.5% bandwidth overhead.

Software Implementation. As a proof of concept, we implemented Alg. 1 in Click, configured an eight-core Intel Nehalem server as a standard IPv4 router, and fed to it a real trace. Then we measured the router's performance with and without running Alg. 1 and saw no difference (in both cases, the server routed 25Gbps). This is not surprising, given that, *when fed realistic traffic*, a Nehalem server is bottlenecked at the I/O [6], whereas our algorithm burdens the CPU.

In our implementation, we computed the $PathID$ of each packet as the concatenation of its source and destination origin prefixes. We implemented the $MarkerID$ and $PacketID$ functions using the "Bob" hash function with different seeds, because it has been shown to work well with Internet traffic [15]. Given that the CAIDA traces do not include the full payload of the captured packets, we applied the two functions only to the IP header (modulo the TTL field) and the small portion of the payload that *is* included—typically 20 bytes of TCP headers. Our results show that this is sufficient, i.e., our implementation indeed collects a random sample from each path, with the sampling rate given by Eq. 1.

7. RELATED WORK

The idea of delayed disclosure of a secret—the sampling seed—has appeared before in networked systems. In the closest related work (which was developed in parallel with our own) Zhang et al. [18] describe a taxonomy of schemes that enable a trusted source/destination pair to identify on-path network adversaries that are maliciously dropping packets. In one of these schemes (PAAI-1), delayed disclosure comes in the form of an explicit request from the source to all the nodes on the path identifying a packet that should be acknowledged. First, that work targets a stronger adversarial model (adversaries who may modify or inject packets) but relies on stronger assumptions as well: symmetric traffic paths and application-layer processing of all receipts by all nodes on a path (onion cryptography). In contrast, Network Confessional makes no claim about the path traversed by receipts to collectors and requires processing only by the issuer of a receipt. Second, PAAI-1 requires that the source generates explicit signaling in addition to normal traffic and that all nodes implement fine-granularity, per-packet timers. In contrast, Network Confessional requires no explicit signaling—using instead later traffic to derive late-disclosed secrets—and a common circular buffer per node without any associated timers. At a higher level, the work by Zhang et al. concerns a usage model that requires end points (e.g., the source and the destination) to be intimately involved implementing functionality such as end-to-end receipts, whereas our approach could be implemented locally, only within a short sub-path of an end-to-end path; and all domains must participate equally to the monitoring scheme, whereas local tunability is an explicit, fundamental requirement of Network Confessional.

The Packet Obituaries protocol [3] and the fault-localization protocols from [9] inform traffic sources where individual packets get lost or corrupted. AudIt provides source domains with similar *per-TCP-flow* information [4]. Network Confessional is similar to these protocols in that it relies on in-path elements collecting and exporting traffic statistics; it also borrows the concept of report consistency from AudIt. However, unlike these protocols, Network Confessional avoids the overheads necessary for collecting and propagating per-packet or per-flow state, while maintaining the verifiability property.

In Trajectory Sampling, routers within an ISP sample packets using a hash function and record their digests, with the purpose of inferring the internal paths (sequences of routers) followed by packets [7]. The Lossy Difference Aggregator enables two monitoring points to measure the loss and average delay between them by maintaining packet counts and average timestamps for packet aggregates [13]. The "Secure Sketch" technique from [10] enables Alice and Bob to detect when the packets they exchange are lost, delayed, or modified beyond a certain level. All three protocols are relevant to our work, in the sense that they measure network performance, but, as explained in Section 2, none of them could provide the properties necessary in our context.

Finally, Network Confessional can be viewed as a "performance accountability mechanism," which holds domains accountable for their performance. An economic analysis has showed that such a performance accountability mechanism would foster ISP competition and innovation [14].

8. CONCLUSIONS

We have presented Network Confessional, a system by which a network monitor can estimate the loss and delay performance of network domains. Each participating domain produces receipts for a small sample of the packets it observes. A domain cannot treat sampled packets preferentially, because the sampling function is keyed on future traffic, which means that a node learns whether it will have to sample a packet, *after* it has forwarded the packet. Moreover, a domain cannot lie about what it did to each sampled packet, because domains that are on the same path are expected to sample the same packets; as a result, to lie about its performance with respect to a packet, a domain would have to implicate one of its neighbors and be exposed to that neighbor as a cheater. Running Network Confessional does not require any form of coordination between different network domains. Deployment comes at the cost of (modest) new functionality at domain boundaries—the capability to produce receipts, which requires computing simple hash functions per observed packet. Producing these receipts requires up to 4 SRAM chips per OC-192 network interface—a reasonable requirement given that such interfaces already come with hundreds of MB of expensive memory.

Acknowledgments. We would like to thank John Byers, Marco Canini, Olivier Crameri, Mihai Dobrescu, Denisa Ghita, Gianluca Iannaccone, Ming Iu, Jean-Philippe Martin-Flatin, Sylvia Ratnasamy, Vyas Sekar, Nina Taft, and Willy Zwaenepoel for their invaluable help.

9. REFERENCES

[1] USNO GPS Time Transfer. `http://tycho.usno.navy.mil/gpstt.html`.

[2] BGP Table Data. `http://bgp.potaroo.net/as6447`, October 2009.

[3] K. Argyraki, P. Maniatis, D. R. Cheriton, and S. Shenker. Providing Packet Obituaries. In *Proceedings of the ACM Workshop on Hot Topics in Networking (HotNets)*, November 2004.

[4] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and Delay Accountability for the Internet. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP)*, October 2007.

[5] J. Burbank, W. Kasch, J. Martin, and D. Mills. Network Time Protocol Version 4 Protocol and Algorithms Specification. `http://tools.ietf.org/html/draft-ietf-ntp-ntpv4-proto-06`, May 2007.

[6] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, October 2009.

[7] N. Duffield and M. Grossglauser. Trajectory Sampling for Direct Traffic Observation. *IEEE/ACM Transactions on Networking*, 9(3):280–292, June 2001.

[8] L. Gharai, C. Perkins, and T. Lehman. Packet reordering, high speed networks and transport protocol performance. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, October 2004.

[9] S. Goldberg, D. Xiao, B. Barak, and J. Rexford. A Cryptographic Study of Secure Internet Measurement. Technical Report TR-783-07, Princeton University, May 2007.

[10] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-Quality Monitoring in the Presence of Adversaries. In *Proceedings of the ACM SIGMETRICS Conference*, June 2008.

[11] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical Accountability for Distributed Systems. In *Proceedings of ACM SOSP*, October 2007.

[12] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical Accountability for Distributed Systems. Technical report, October 2007. Available at `http://www.cis.upenn.edu/~ahae/papers/peerreview-tr2.pdf`.

[13] R. R. Kompella, K. Levchenko, A. C. Snoeren, and G. Varghese. Every Microsecond Counts: Tracking Fine-Grain Latencies with a Lossy Difference Aggregator. In *Proceedings of the ACM SIGCOMM Conference*, August 2009.

[14] P. Laskowski and J. Chuang. Network Monitors and Contracting Systems. In *Proceedings of the ACM SIGCOMM Conference*, September 2006.

[15] M. Molina, S. Niccolini, and N. G. Duffield. A Comparative Experimental Study of Hash Functions Applied to Packet Sampling. In *Proceedings of International Teletraffic Congress (ITC)*, September 2005.

[16] P. Phaal and S. Panchen. Sampling Basics. `http://www.sflow.org/packetSamplingBasics/index.htm`.

[17] J. Sommers, P. Barford, N. Duffied, and A. Ron. Accurate and Efficient SLA Compliance Monitoring. In *Proceedings of the ACM SIGCOMM Conference*, August 2007.

[18] X. Zhang, A. Jain, and A. Perrig. Packet-dropping Adversary Identification for Data Plane Security. In *Proceedings of the ACM CoNext Conference*, December 2008.

APPENDIX

A. PROOF OF LEMMA 4.2

We consider a period of time during which each of nodes 4 and 5 from Fig. 1 uses a constant buffer size and sampling threshold; β is the smallest of the two buffer sizes used by the two nodes, and σ is the smallest of the two sampling thresholds.

A.1 Assumptions

We assume that packet loss and reordering are independent and identically distributed Bernoulli processes, and that reordering can only happen between packets that are observed relatively close to one another.

More specifically, we assume that: (1) Each packet observed at node 4 is lost between nodes 4 and 5 with probability λ, independently from any other packet. (2) A packet p that is observed at node 4 between markers m_1 and m_2, and is not lost between nodes 4 and 5, is reordered with either m_1 or m_2 with probability ρ, independently from any other packet. (3) Moreover, p can only be reordered with m_1 or m_2, not with any other previous or future markers.

These is a simple model, and there is no guarantee that it accurately describes Internet conditions. However, we are not aware of any more sophisticated models that have been shown to be more accurate, either. Note that we use this model only to derive an estimate of the accuracy of our loss estimation—it does not affect Network Confessional in any other way.

A.2 Definitions

We define the following symbols:

Number of potentially sampled packets N. A packet p counts toward N when: (i) p was observed at node 4 before a marker m, which was also observed at node 5. (ii) The two markers observed at node 4 right before and right after m were also observed at node 5 in the same order. I.e., if node 4 observed packet sequence $\langle m_p, ...p, ...m, ...m_n \rangle$, node 5 observed packet sequence $\langle m_p, ...m, ...m_n \rangle$, where m_p, m, and m_n are all markers. The reason for defining N in this way will become apparent below.

Number of commonly sampled packets K. The number of potentially sampled packets that are sampled by node 4, and, unless they are lost or reordered with a marker, are also sampled by node 5. A potentially sampled packet that is not lost or reordered with a marker is sampled by both nodes with probability π_s, given by Eq. 1. Hence, K is a random variable of binomial distribution, with expected value $E(K) = N\ \pi_s$.

Number of lost sampled packets k_l. The number of commonly sampled packets that are lost between the two nodes and, as a result, not sampled by node 5. More specifically, a potentially sampled packet p belongs to this group of packets when:

1. p is commonly sampled. The probability of this event is π_s.
2. p is lost between nodes 4 and 5. The probability of this event is λ.

Given Assumption (1), k_l is a random variable of binomial distribution, with expected value $E(k_l) = N\ \pi_s\ \lambda$.

Number of downgraded packets k_r. The number of commonly sampled packets that are reordered with a marker and, as a result, not sampled by node 5. More specifically, a potentially sampled packet p belongs to this group of packets when:

1. p is observed at node 4 between markers m_1 and m_2 and is commonly sampled.
2. p is not lost between nodes 4 and 5.

3. p is reordered with either m_1 or m_2 between nodes 4 and 5. The probability of this event is ρ.

4. p is not sampled by node 5.

Given Assumptions (1) and (2), k_r is a random variable of binomial distribution, with expected value $E(k_r) = N\ \pi_s\ \pi_r$, where $\pi_r = (1-\lambda)\ \rho\ (1-\pi_s)$.

Number of upgraded packets $\bar{k_r}$. The number of potentially sampled packets that are *not* sampled by node 4, but, due to packet reordering, *are* sampled by node 5. More specifically, a potentially sampled packet p belongs to this group of packets when:

1. p is observed at node 4 between markers m_1 and m_2 and is *not* commonly sampled.

2. p is not lost between nodes 4 and 5.

3. p is reordered with either m_1 or m_2 between nodes 4 and 5.

4. p is sampled by node 5.

Given Assumptions (1) and (2), $\bar{k_r}$ is a random variable of binomial distribution, with expected value $E(\bar{k_r}) = N\ (1-\pi_s)\ \bar{\pi_r}$, where $\bar{\pi_r} = (1-\lambda)\ \rho\ \pi_s$.

Number of missed sampled packets k. These are the commonly sampled packets that are not sampled by node 5, either because they are lost or because they are downgraded. I.e., $k = k_l - k_r$.

Loss estimate λ^*. Of the above symbols, the receipt collector can compute K, k, and $\bar{k_r}$. It estimates λ as

$$\lambda^* = \frac{k - k_l}{K}$$

At this point, we can explain why N was defined as it was: because that definition, in combination with Assumption (3), enabled us to define k_r and $\bar{k_r}$ such that (i) they have the same probability mass function, and (ii) the receipt collector can compute $\bar{k_r}$. We clarify point (ii): Consider an upgraded packet p that is sampled by node 5 based on a marker m; the previous and next markers observed at node 5 are m_p and m_n. To determine that p is an upgraded packet, the receipt collector must first determine whether p is a potentially sampled packet, i.e., whether p was observed at node 4 before a marker m' that was also observed at node 5. Since p was not sampled by node 4, the receipt collector does not know before which marker p was observed at node 4. However, given Assumption (3), it knows that p was observed either before m_p or before m_n at node 4; both of these packets were also observed at node 5, hence, p is a potentially sampled packet.

A.3 Expected Value

The expected value of the estimate λ^* is equal to:

$$\begin{aligned} E(\lambda^*) &= E\left(\frac{k - \bar{k_r}}{K}\right) \\ &= E\left(\frac{k_l + k_r - \bar{k_r}}{K}\right) \\ &= E\left(\frac{k_l}{K}\right) + E\left(\frac{k_r}{K}\right) - E\left(\frac{\bar{k_r}}{K}\right). \end{aligned} \tag{4}$$

We first compute $E\left(\frac{k_l}{K}\right)$:

$$\begin{aligned} E\left(\frac{k_l}{K}\right) &= \sum_{K=0}^{N} \sum_{k_l=0}^{K} \frac{k_l}{K}\ pmf(k_l, K) \\ &= \sum_{K=0}^{N} \sum_{k_l=0}^{K} \frac{k_l}{K}\ pmf(k_l|K) \cdot pmf(K) \\ &= \sum_{K=0}^{N} \frac{1}{K}\ pmf(K) \sum_{k_l=0}^{K} k_l\ pmf(k_l|K) \\ &= \sum_{K=0}^{N} \frac{1}{K}\ pmf(K) \cdot E(k_l|K) \\ &= \sum_{K=0}^{N} \frac{1}{K}\ pmf(K) \cdot K\ \lambda \\ &= \sum_{K=0}^{N} pmf(K) \cdot \lambda \\ &= \lambda \sum_{K=0}^{N} pmf(K) \\ &= \lambda. \end{aligned} \tag{5}$$

In a similar way, we can show that:

$$E\left(\frac{k_r}{K}\right) = E\left(\frac{\bar{k_r}}{K}\right) = \pi_r. \tag{6}$$

Combining Eqs. 4, 5 and 6, we get:

$$E(\lambda^*) = \lambda. \tag{7}$$

A.4 Relative Standard Deviation

The variance of λ^* is equal to:

$$\begin{aligned}
&Var(\lambda^*) = \\
&E((\lambda^*)^2) - (E(\lambda^*))^2 = \\
&E\left(\left(\frac{k - \bar{k_r}}{K}\right)^2\right) - \lambda^2 = \\
&E\left(\left(\frac{k_l + k_r - \bar{k_r}}{K}\right)^2\right) - \lambda^2 = \\
&E\left(\frac{k_l^2 + k_r^2 + \bar{k_r}^2 + 2k_l - 2k_l - 2k_r}{K^2}\right) - \lambda^2 = \\
&E\left(\frac{k_l^2}{K^2}\right) + E\left(\frac{k_r^2}{K^2}\right) + E\left(\frac{\bar{k_r}^2}{K^2}\right) \\
&+ 2\ E\left(\frac{k_l\ k_r}{K^2}\right) - 2\ E\left(\frac{k_l\ \bar{k_r}}{K^2}\right) - 2\ E\left(\frac{k_r\ \bar{k_r}}{K^2}\right) \\
&- \lambda^2. \qquad (8)
\end{aligned}$$

We first compute $E\left(\frac{k_l^2}{K^2}\right)$:

$$\begin{aligned}
E\left(\frac{k_l^2}{K^2}\right) &= \sum_{K=0}^{N} \sum_{k_l=0}^{K} \frac{k_l^2}{K^2}\ pmf(K, k_l) \\
&= \sum_{K=0}^{N} \sum_{k_l=0}^{K} \frac{k_l^2}{K^2}\ pmf(k_l|K) \cdot pmf(K) \\
&= \sum_{K=0}^{N} \frac{1}{K^2}\ pmf(K) \sum_{k_l=0}^{K} k_l^2\ pmf(k_l|K) \\
&= \sum_{K=0}^{N} \frac{1}{K^2}\ pmf(K)\ E[{k_l}^2|K] \\
&= \sum_{K=0}^{N} \frac{1}{K^2}\ pmf(K)\ \left(Var[k_l|K] + (E[k_l|K])^2\right) \\
&= \sum_{K=0}^{N} \frac{1}{K^2}\ pmf(K)\ \left(Var[k_l|K] + \left(K \cdot E\left[\frac{k_l}{K}|K\right]\right)^2\right) \\
&= \sum_{K=0}^{N} \frac{1}{K^2}\ pmf(K)\ \left(Var[k_l|K] + K^2\lambda^2\right) \\
&= \sum_{K=0}^{N} \frac{1}{K^2}\ pmf(K)\ \left(K\lambda(1-\lambda) + K^2\lambda^2\right) \\
&= \lambda^2 \left(\sum_{K=0}^{N} pmf(K)\right) + \\
&\quad \lambda(1-\lambda)\left(\sum_{K=0}^{N} \frac{1}{K}\ pmf(K)\right) \\
&= \lambda^2 + \lambda(1-\lambda)\left(\sum_{K=0}^{N} \frac{1}{K}\ pmf(K)\right) \\
&= \lambda^2 + \lambda(1-\lambda)E\left[\frac{1}{K}\right] \\
&= \lambda^2 + \frac{\lambda(1-\lambda)}{E[K]} \\
&= \lambda^2 + \frac{\lambda(1-\lambda)}{N\pi_s}. \qquad (9)
\end{aligned}$$

In a similar way, we can show that:

$$\begin{aligned}
E\left(\frac{k_r^2}{K^2}\right) &= E\left(\frac{\bar{k_r}^2}{K^2}\right) \\
&= \pi_r^2 + \frac{\pi_r(1-\pi_r)}{N\pi_s}. \qquad (10)
\end{aligned}$$

Next, we compute $E\left(\frac{k_r\ \bar{k_r}}{K^2}\right)$:

$$\begin{aligned}
E\left(\frac{k_r\ \bar{k_r}}{K^2}\right) &= \sum_{K=0}^{N}\sum_{k_r=0}^{K}\sum_{\bar{k_r}=0}^{N-K}\ \frac{k_r\ \bar{k_r}}{K^2}\ \ pmf(K,k_r,\bar{k_r}) \\
&= \sum_{K=0}^{N}\sum_{k_r=0}^{K}\sum_{\bar{k_r}=0}^{N-K}\ \frac{k_r\ \bar{k_r}}{K^2}\ \ pmf(\bar{k_r}|k_r,K)\cdot pmf(k_r|K)\cdot pmf(K) \\
&= \sum_{K=0}^{N}\ \frac{1}{K^2}\ \ pmf(K)\sum_{k_r=0}^{K}\ k_r\ \ pmf(k_r|K)\ \sum_{\bar{k_r}=0}^{N-K}\ \bar{k_r}\ \ pmf(\bar{k_r}|k_r,K) \\
&= \sum_{K=0}^{N}\ \frac{1}{K^2}\ \ pmf(K)\cdot E[k_r|K]\ \sum_{\bar{k_r}=0}^{N-K}\ \bar{k_r}\ \ pmf(\bar{k_r}|k_r,K) \\
&= \sum_{K=0}^{N}\ \frac{1}{K^2}\ \ pmf(K)\cdot E[k_r|K]\cdot E[\bar{k_r}|k_r,K] \\
&= \sum_{K=0}^{N}\ \frac{1}{K^2}\ \ pmf(K)\cdot E[k_r|K]\cdot E[\bar{k_r}|K]\quad \text{(because the underlying processes generating } \bar{k_r} \text{ and } k_r \text{ are indep.)} \\
&= \sum_{K=0}^{N}\ \frac{1}{K^2}\ \ pmf(K)\cdot(K\ \pi_r)\cdot((N-K)\ \bar{\pi_r}) \\
&= \sum_{K=0}^{N}\ \frac{1}{K}\ \ pmf(K)\cdot\ \pi_r\cdot((N-K)\ \bar{\pi_r}) \\
&= \pi_r\ \bar{\pi_r}\ \sum_{K=0}^{N}\ \frac{N-K}{K}\ \ pmf(K) \\
&= \pi_r\ \bar{\pi_r}\ \left(\sum_{K=0}^{N}\ \frac{N}{K}\ \ pmf(K)-\sum_{K=0}^{N}\ \ pmf(K)\right) \\
&= \pi_r\ \bar{\pi_r}\ \left(N\cdot\sum_{K=0}^{N}\frac{1}{K}pmf(K)\ -\ 1\right) \\
&= \pi_r\ \bar{\pi_r}\ \left(N\cdot E\left[\frac{1}{K}\right]-1\right) \\
&= \pi_r\ \bar{\pi_r}\ \left(\frac{N}{E[K]}-1\right) \\
&= \pi_r\ \bar{\pi_r}\ \left(\frac{N}{N\pi_s}-1\right) \\
&= \pi_r\cdot\bar{\pi_r}\cdot\frac{1-\pi_s}{\pi_s} \\
&= \pi_r\cdot(1-\lambda)\rho\pi_s\cdot\frac{1-\pi_s}{\pi_s} \\
&= \pi_r\cdot(1-\lambda)\rho(1-\pi_s) \\
&= \pi_r^2. \qquad (11)
\end{aligned}$$

In a similar way, we can show that:

$$E\left(\frac{k_l\ k_r}{K^2}\right) = E\left(\frac{k_l\ \bar{k_r}}{K^2}\right). \tag{12}$$

To compute the relative standard deviation, we first have

$$\begin{aligned}
Var(\lambda^*) &= E\left(\frac{k_l^2}{K^2}\right) + E\left(\frac{k_r^2}{K^2}\right) + E\left(\frac{\bar{k_r}^2}{K^2}\right) + 2\ E\left(\frac{k_l\ k_r}{K^2}\right) - 2\ E\left(\frac{k_l\ \bar{k_r}}{K^2}\right) - 2\ E\left(\frac{k_r\ \bar{k_r}}{K^2}\right) - \lambda^2 \quad \text{(as per Eq. 8)}\\
&= E\left(\frac{k_l^2}{K^2}\right) + E\left(\frac{k_r^2}{K^2}\right) + E\left(\frac{\bar{k_r}^2}{K^2}\right) - 2\ E\left(\frac{k_r\ \bar{k_r}}{K^2}\right) - \lambda^2 \quad \text{(as per Eq. 12)}\\
&= \lambda^2 + \frac{\lambda(1-\lambda)}{N\pi_s} + E\left(\frac{k_r^2}{K^2}\right) + E\left(\frac{\bar{k_r}^2}{K^2}\right) - 2\ E\left(\frac{k_r\ \bar{k_r}}{K^2}\right) - \lambda^2 \quad \text{(as per Eq. 9)}\\
&= \frac{\lambda(1-\lambda)}{N\pi_s} + E\left(\frac{k_r^2}{K^2}\right) + E\left(\frac{\bar{k_r}^2}{K^2}\right) - 2\ E\left(\frac{k_r\ \bar{k_r}}{K^2}\right)\\
&= \frac{\lambda(1-\lambda)}{N\pi_s} + 2\cdot\left(\pi_r^2 + \frac{\pi_r(1-\pi_r)}{N\pi_s}\right) - 2\ E\left(\frac{k_r\ \bar{k_r}}{K^2}\right) \quad \text{(as per Eq. 10)}\\
&= \frac{\lambda(1-\lambda)}{N\pi_s} + 2\cdot\left(\pi_r^2 + \frac{\pi_r(1-\pi_r)}{N\pi_s}\right) - 2\pi_r^2 \quad \text{(as per Eq. 11)}\\
&= \frac{\lambda(1-\lambda)}{N\pi_s} + 2\cdot\left(\pi_r^2 + \frac{\pi_r(1-\pi_r)}{N\pi_s}\right) - 2\pi_r^2\\
&= \frac{\lambda(1-\lambda)}{N\pi_s} + 2\cdot\frac{\pi_r(1-\pi_r)}{N\pi_s}
\end{aligned} \tag{13}$$

The relative standard deviation is therefore

$$\frac{\sigma(\lambda^*)}{\lambda} = \frac{\sqrt{Var(\lambda^*)}}{\lambda} = \sqrt{\frac{Var(\lambda^*)}{\lambda^2}} = \sqrt{\frac{1-\lambda}{N\ \pi_s\ \lambda} + \frac{2\ \pi_r(1-\pi_r)}{N\ \pi_s\ \lambda^2}}. \tag{14}$$

Eqs. 7 and 14 prove Lemma 3.2.