Verifiable Network-Performance Measurements

Katerina Argyraki
EPFL, Switzerland

Petros Maniatis
Intel Labs Berkeley, USA

Ankit Singla*
UIUC, USA

ABSTRACT

In the current Internet, there is no clean way for affected parties to react to poor forwarding performance: to detect and assess Service Level Agreement (SLA) violations by a contractual partner, a domain must resort to ad-hoc monitoring using probes. Instead, we propose *Network Confessional*, a new, systematic approach to the problem of forwarding-performance verification. Our system relies on voluntary reporting, allowing each network domain to disclose its loss and delay performance to its customers and peers and, potentially, a regulator. Most importantly, it enables *verifiable* performance measurements, i.e., domains cannot abuse it to significantly exaggerate their performance. Finally, our system is *tunable*, allowing each participating domain to determine how many resources to devote to it independently (i.e., without any inter-domain coordination), exposing a controllable trade-off between performance-verification quality and resource consumption. Our system comes at the cost of deploying modest functionality at the participating domains' border routers; we show that it requires reasonable resources, well within modern network capabilities.

1. INTRODUCTION

The lack of a systematic method for estimating the performance of Internet service providers (ISPs) is a well known problem: when an ISP does not perform as expected, there is no clean way for the affected parties to detect the problem so they can debug it, ask for compensation if a Service-Level Agreement (SLA) has been violated, or simply learn from it (e.g., re-assess a peering agreement with an underperforming neighbor). This lack of information makes network debugging difficult and slow, even leading ISPs to deny their failures to their customers and peers, pointing fingers at one another. One could attribute this situation to the best-effort nature of the Internet which, by definition, provides no a-priori guarantees. Yet that is no reason not to expect useful, after-the-fact information about ISP performance—actually, it makes perfect sense to expect such information in a best-effort environment like the Internet, where communication quality often relies on quick failure detection and on choosing the right providers and peers.

Since ISPs offer no explicit interface for their customers and peers to verify their performance, the latter can only resort to probing tools like traceroute or other active measurements. Moreover, researchers have recently started to combine probing from multiple vantage points (e.g., PlanetLab nodes) to gain information about ISP performance that would not be accessible through simple probing [15, 16]. This information is typically extracted from channels with a different purpose (e.g., ICMP traffic), because probing mechanisms are designed under the assumption that ISPs would never freely provide honest information about their performance.

But what if an ISP is required, e.g., by government regulation, to expose information on how it treats different traffic flows? The UK telecommunications regulator already took a first step in 2009 by publicly disclosing information on ISP performance [4], and there is ongoing debate regarding the extent to which ISPs' traffic handling should be regulated [5]. Yet experience says that, once ISPs know that their performance is being measured in a certain way, they find ways to game the measurement process. Hence, we believe that this is the right time to discuss the design and implementation of an interface through which an ISP can provide accurate and verifiable information on its performance, while consuming a reasonable, tunable amount of resources for this purpose.

Moreover, given an either-or choice, an ISP may prefer to expose itself information on its performance rather than have its performance evaluated by untrusted entities, through potentially inaccurate mechanisms. Probing or other edge-based "black-box" mechanisms typically run on coalitions of end systems like PlanetLab; the ISP has no reason to trust these, and they can provide no guarantee for the accuracy of their measurements. If an ISP's performance is to be talked about anyway, an accurate, trusted self-reporting system may be preferable to the ISP, because, at least, it provides the ISP with control over the quality and quantity of the information that is revealed about its business.

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

ACM CoNEXT 2010, November 30 – December 3 2010, Philadelphia, USA.

Finally, ISPs often need to exchange performance information anyway with their customers and peers, in order to handle customer complaints. When a customer calls her ISP to complain that she cannot reach a certain destination, the ISP needs to know whether the problem lies in its own local network, the customer's network, the network of the peer that is handling traffic to that destination, or the destination's network—because each of these cases warrants a different response. Today, this information is acquired by ISP operators in a reactive, ad-hoc manner, which means that it takes time to resolve each complaint, potentially leaving customers dissatisfied. It makes sense that an ISP would prefer to collaborate with its customers and peers and willingly exchange troubleshooting reports with them, provided that it can trust these reports to be accurate and honest.

In the rest of the paper, we describe *Network Confessional*, a protocol that enables verifiable network-performance measurements. Each ISP that runs this protocol produces traffic receipts on sampled packets at its entry and exit points (i.e., border routers), exchanges these with the other networks that observe its traffic, and, potentially, makes them available to a verifier such as a regulator. These receipts are specially crafted, so that: (1) they enable accurate estimation of ISP performance, without revealing any information about the internal structure or routing policies of ISPs beyond what is already publicly available through BGP routing tables; (2) ISPs cannot produce fake receipts to significantly exaggerate their performance; and (3) each ISP can choose how many resources to devote to receipt generation independently from others, yet in a way that does not compromise the verifiability of the derived measurements. These features come at the cost of deploying new functionality at the participating domains' border routers, but we show that that requires reasonable resources and readily available hardware (we describe one implementation that requires a single 2 MB TCAM chip per router linecard).

We start, in Section 2, with our problem statement, threat model, and assumptions. Then we explain, in Section 3, why straightforward extensions of existing techniques fail to provide an adequate solution. We describe Network Confessional in two parts: first, under certain simplifying assumptions in Section 4, then without these assumptions in Section 5. We show how to deploy it using readily available hardware in Section 6 and experimentally evaluate the quality of information it provides in Section 7. We discuss related work in Section 8 and conclude in Section 9.

2. SETUP

Informally, we want to design a measurement protocol such that (1) domains can accurately estimate the performance of their neighbors, (2) domains cannot bias the measurement process to their advantage without being detected, and (3) they need a reasonable, tunable amount of resources to collect and exchange the measurements.

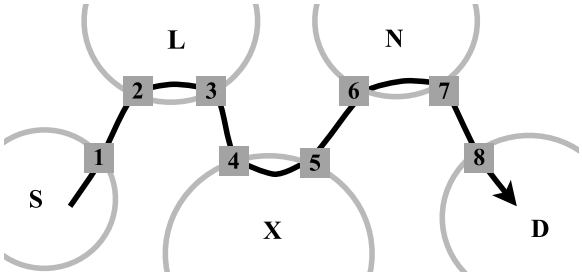

Figure 1: Circles represent administrative domains. The numbered boxes represent border routers.

2.1 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).

A *packet stream* observed at node i is a time series, where each element corresponds to a packet and the time at which the packet was observed at node i.

2.2 Problem Statement

We consider a set of paths. 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.

Each node i on path k observes a packet stream P_i^k. Each node j that comes after i on path k, observes a packet stream $P_j^k = \mathcal{T}_{ij}^k(P_i^k)$, where $\mathcal{T}_{ij}^k$ denotes transformation and may be any combination of packet loss, delay and reordering (but may not involve packet injection or modification). I.e., the packets in P_j^k are always a subset of the packets in P_i^k, and two packets may appear with a different order in the two packet streams. 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.

All the nodes on each path k participate in a measurement protocol, according to which, each node i that observes packet stream P_i^k, computes a set of receipts, $R_i^k = \mathcal{F}_r(P_i^k)$. All receipts generated by all nodes on path k are correctly

delivered to a set of *receipt collectors*, which compute a set of functions on them. The receipt collectors may include any of the domains that observe traffic from path k and/or a regulator. Each receipt collector can compute three functions:

1. For each pair of an input node i and output node j that comes after i on path k, a *loss function* $\mathcal{F}_\lambda(R_i^k, R_j^k)$, which returns an estimate of λ_{ij}^k, and a *delay-quantile function* $\mathcal{F}_\delta(R_i^k, R_j^k, q)$, which returns an estimate of $\delta_{ij}^k(q)$.

2. For each pair of peering nodes i and j on path k, a *verification function* $\mathcal{F}_v(R_i^k, R_j^k)$, which returns "true" or "false." "True" indicates that both peering nodes are running the measurement protocol correctly with respect to the traffic from path k, and the inter-domain link between them is not faulty.

Our threat model is as follows. Node i is *honest* with respect to path k, if it computes R_i^k using the specified $\mathcal{F}_r$. Node i is *lying* with respect to path k, if it computes at least one receipt in R_i^k using an arbitrary $\hat{\mathcal{F}}_r$. Lying nodes can collude and choose their $\hat{\mathcal{F}}_r$ in coordination. A domain is *honest* with respect to path k, if both of its nodes are honest with respect to k, otherwise it is *lying*.

We want to design a measurement protocol (i.e., specify functions $\mathcal{F}_r$ $\mathcal{F}_\lambda$, $\mathcal{F}_\delta$ and $\mathcal{F}_v$), such that the following conditions are met:

1. If input node i and output node j on path k are honest with respect to k, then $|\lambda_{ij}^k - \mathcal{F}_\lambda(R_i^k, R_j^k)| < l_{ij}$ and $|\delta_{ij}^k(q) - \mathcal{F}_\delta(R_i^k, R_j^k, q)| < d_{ij}$ with probability π_{ij}, where l_{ij}, d_{ij} and π_{ij} are configurable parameters, chosen by nodes i and j. Computing R_i^k and R_j^k for all paths k in which nodes i and j participate, requires an amount of memory and computing cycles that depends on l_{ij}, d_{ij}, and π_{ij}.

2. If peering nodes i and j on path k are honest with respect to k, and the inter-domain link between them is not faulty, then $\mathcal{F}_v(R_i^k, R_j^k)$ is "true," otherwise it is "false."

Given that packet streams may be infinite or arbitrarily long, the receipt collectors should be able to compute their functions, and the above properties should hold over fixed time intervals.

For brevity, when it is obvious from the context that we are referring to a particular path, we drop the superscript k from P_i^k, R_i^k, λ_{ij}^k, and δ_{ij}^k.

Discussion. The first condition ensures that a receipt collector can estimate the performance of honest domains with probabilistic guarantees, and that the measurement protocol does not require any per-packet, per-flow or per-path state. The latter is important, because a node may observe hundreds of thousands, perhaps even millions of concurrent flows and paths.

The first condition also ensures that collusion comes at a cost: two adjacent domains are free to collude such that a receipt collector cannot accurately estimate their performance; but, if they do that, one of them will appear to have better performance at the expense of the other. For instance, consider the path depicted in Fig. 1. Suppose that domain X introduces loss λ_{45}, while domain N introduces loss λ_{67}. Suppose that node 5 is lying, such that $\mathcal{F}_\lambda(R_4, R_5) = \lambda_{45} - \Lambda$, where $\Lambda \gg 0$ (i.e., from X's receipts, it looks like its packet loss is significantly lower than it actually is). Finally, suppose that node 6 is also lying, such that $\mathcal{F}_v(R_5, R_6) =$ "true" (i.e., domain N covers X's lie). According to the first condition, as long as nodes 4 and 7 are honest, $\mathcal{F}_\lambda(R_4, R_7) \approx \lambda_{47}$; given that packet loss is additive, this necessarily means that $\mathcal{F}_\lambda(R_6, R_7) \approx \lambda_{67} + \Lambda$, i.e., from N's receipts, it looks like its packet loss is significantly *higher* than it actually is.

The second condition ensures that, if a domain deviates from the protocol, it necessarily implicates one of its neighbors and is exposed to that neighbor as a liar. For instance, suppose that node 5 in Fig. 1 is lying (e.g., to hide the fact that X is dropping packets). If node 6 is honest, that will cause $\mathcal{F}_v(R_5, R_6)$ to return "false," alerting the receipt collector that either 5 or 6 is lying, or the link between them is faulty. In general, a receipt collector cannot know which of these is true, but domain N can: it can debug the inter-domain link, determine that it is functioning correctly, and conclude that X is lying and is implicating N in its lie.

2.3 Assumptions

We make the following assumptions:

(1) There exists a way for a domain to disseminate receipts to any other domain, such that the authenticity and integrity of each received receipt is guaranteed. One way of realizing this assumption would be for each domain to make its receipts available at an administrative web-site and accessible over HTTPS. It is possible to design more efficient dissemination mechanisms, but that is outside the scope of this paper.

(2) Each domain has some network equipment (routers or other middleboxes) that can perform at wire speed simple per-packet operations. Those 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. This assumption is in line with current trends in production routers, as well as the increasing focus of academia and industry on programmable routers and switches [10, 19].

(3) Nodes (whether honest or lying) do not apply any transformation to the observed packet stream other than packet loss, delay, or reordering. In particular, they do not inject new packets or modify observed packets. To the best of our knowledge, packet injection and modification is further from current ISP practices (than introducing loss or unpredictable delay and denying performance problems), and we defer dealing with this behavior to future work.

3. WHY A NEW PROTOCOL

There already exist many good techniques for measuring network performance [11, 14, 17, 22]. So, 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 [6]. $\mathcal{F}_r$ produces a receipt for every observed packet, which consists of a *digest* for the corresponding packet and the *timestamp* for when the packet was observed. $\mathcal{F}_\lambda(R_i^k, R_j^k)$ and $\mathcal{F}_\delta(R_i^k, R_j^k, q)$ are straightforward—the former counts how many packets from path k were observed at node i versus node j, while the latter relies on comparing the timestamps recorded for the same packet at node i versus node j. $\mathcal{F}_v(R_i^k, R_j^k)$ returns "false" if there exists at least one packet from path k such that: i produced a receipt for it but j did not, or 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.

This protocol fails to meet our first 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 a simple combination of Packet Obituaries and Trajectory Sampling [11] (POTS, for brevity): $\mathcal{F}_r$ 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 produced for it (note that, since all nodes use the same sampling function, they all sample the same packets). $\mathcal{F}_\lambda$ and $\mathcal{F}_\delta$ can be any functions that estimate the loss and delay experienced by all packets, based on the loss and delay experienced by a representative subset of sampled packets [22]. $\mathcal{F}_v$ is the same as in Packet Obituaries+.

This protocol fails to meet our first condition: it is possible that all nodes are honest (i.e., run the protocol as specified), yet $\mathcal{F}_\lambda$ and $\mathcal{F}_\delta$ return arbitrarily inaccurate results. In particular, each input node can engage in the following behavior: for each observed packet, run $\mathcal{F}_r$, 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.

Uncoordinated Trajectory Sampling. Since the problem with POTS is that all domains sample the same packets (hence have all an incentive to bias the same traffic), one approach would be to make each domain sample different packets. Hence, we consider the following variation on POTS. $\mathcal{F}_r$ works as in POTS, but the hash function is different for each domain, such that each domain samples a different subset of the packets. A node/domain is *honest*, if it uses $\mathcal{F}_r$ to produce its receipts *and* does not treat sampled packets differently than the rest of the traffic, and *lying*, otherwise. $\mathcal{F}_\lambda$ and $\mathcal{F}_\delta$ are the same as in POTS. $\mathcal{F}_v(R_i^k, R_j^k)$ returns "true," if nodes i and j generate consistent measurements with the two nodes that come before i and the two nodes that come after j on path k. For instance, in the path depicted in Fig. 1, $\mathcal{F}_v(R_3, R_4)$ returns "true," if $\mathcal{F}_\lambda(R_1, R_4) \approx \mathcal{F}_\lambda(R_2, R_3)$ and $\mathcal{F}_\lambda(R_3, R_6) \approx \mathcal{F}_\lambda(R_4, R_5)$ (and the same for delay). If it returns "false," e.g., because $\mathcal{F}_\lambda(R_3, R_6) > \mathcal{F}_\lambda(R_4, R_5)$, then: either the link between L and X, or the link between X and N, or both are faulty; or domain X is lying to exaggerate its performance; or domain L and/or domain N are lying to misrepresent X's performance. In this fashion, one could argue, we can localize lying (including sampling bias) to a sequence of three suspect domains.

This protocol fails to meet our second condition: Even if domains sample non-overlapping subsets of packets, they can collude, such that all of them treat all subsets of sampled packets preferentially. In this way, all nodes are lying, yet $\mathcal{F}_v$ always returns "true"—and $\mathcal{F}_\lambda$ and $\mathcal{F}_\delta$ return arbitrarily inaccurate results.

Verifiable Aggregation. An alternative to sampling is aggregation: instead of producing receipts for sampled packets, produce receipts for packet aggregates. We could design such an alternative by combining Packet Obituaries with either the Lossy Difference Aggregator [17] or the "Secure Sketch" technique from [14]. We have explored that alternative extensively elsewhere [8]. To summarize, these combinations fail to meet our first condition in two ways. First, they do not provide a delay-quantile function $\mathcal{F}_\delta$ (only statistics on average delay). Most importantly, computing their $\mathcal{F}_r$ requires maintaining per-path state.

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 first condition, and with per-domain tunability.

4. 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.

4.1 Receipt Generation

Each node samples a subset of the packets it observes and generates a receipt for each sampled packet; each receipt is made available to all nodes on the path of the corresponding packet and, potentially, a verifier such as a regulator. 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. A naïve 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 [23]. That, however, would essentially require every source domain to set up virtual circuits along all Internet paths that observe its traffic. 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, 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).

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 ← ⟨PathID(p), PacketID(p), Time⟩
2: if MarkerID(p) < μ then
3:    for all T in Buffer do
4:       if T.PathID = T_p.PathID then
5:          if Hash(T.PacketID, T_p.PacketID) < σ 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
```

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

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

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 $\frac{\alpha\mu}{\mathcal{M}}$. 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 $\frac{\alpha\mu}{\mathcal{M}}$. It follows that the probability that $Distance(p, m) < \beta$ is equal to the cumulative distribution function (CDF) of the geometric distribution, i.e., $1 - \left(1 - \frac{\alpha\mu}{\mathcal{M}}\right)^{\beta}$. Next, we com-

pute 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 $\left(1-\left(1-\frac{\alpha\mu}{\mathcal{M}}\right)^{\beta}\right)\cdot\frac{\sigma}{\mathcal{S}}$. □

As we explain in Section 6, in practice, $\beta \gg \frac{\mathcal{M}}{\alpha\mu}$ for any $\alpha > 0.01$. As long as traffic from a certain path exceeds 1% of the overall traffic observed at a node, $\pi_s \approx \frac{\sigma}{\mathcal{S}}$, i.e., β does not affect the sampling rate for that path.

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

4.2 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. The key element that enables this property is the inequality in line 5 of Alg. 1. Consider nodes 1 and 2, with the same buffer size, but 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 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.

Moreover, each node chooses its own buffer size β. At the same time, given any number of nodes and their buffer sizes, we maximize the number of packets that are commonly sampled by all nodes. To illustrate, we consider nodes 1 and 2, with the same sampling threshold, but different buffer sizes β_1 and $\beta_2 > \beta_1$. As in the previous paragraph, 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 observed after p. Since node 1 samples p, this necessarily means that $Distance(p, m) < \beta_1 < \beta_2$, which means that node 2 also samples p.

So, even though each node chooses its σ and β independently, if there is no packet loss or reordering, a node with bigger σ and β will sample at least all the packets sampled by a node with smaller σ and β.

4.3 Receipt-based Statistics

We now consider a receipt collector that collects receipts from the nodes in Fig. 1 and describe how it computes and verifies the performance of domain X.

Loss Function. 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 receipt collector considers the receipts R_4 and R_5 generated by the two nodes during a given time period. By looking at the $ReporterConfig$ values of these receipts, it divides the time period into sub-periods, such that, throughout each sub-period, each node used a constant buffer size and sampling threshold. For each sub-period, 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 $Distance(p, m) < \beta_{min}$ and $Hash(PacketID(p), PacketID(m)) < \sigma_{min}$, where β_{min} and σ_{min} are the smaller buffer size and 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 $\mathcal{F}_\lambda(R_4, R_5) = \frac{k}{K}$.

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, and $k_r = k - k_l$ 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 [12], 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 $Distance(p, m) < \beta_{min}$ and $Hash(PacketID(p), PacketID(m)) < \sigma_{min}$. Hence, the receipt collector computes $\bar{k_r}$ and estimates the loss rate between the two nodes as $\mathcal{F}_\lambda(R_4, R_5) = \frac{k-\bar{k_r}}{K}$, i.e., it approximates k_r with $\bar{k_r}$.

Lemma 4.2. *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 1.

PROOF. In our technical report [8]. □

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 1: Parameters for Lemma 4.2.

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 [21].

Lemma 4.2 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-Quantile Function. The receipt collector considers all the receipts generated by nodes 4 and 5 during a given time period. By looking at the $PacketID$ 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 [22]. 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 Function. The receipt collector considers all the receipts generated by each pair of peering nodes i and j during a given time period. 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 Function" paragraph above). $\mathcal{F}_v(R_i, R_j)$ returns "true" when all of the following hold for all packets p that belong to this set: (1) Either both nodes i and j 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.

In Section 2.2, we stated that we wanted function $\mathcal{F}_v(R_i, R_j)$ to return "true," if both nodes i and j run the protocol correctly and the link between them is not faulty, otherwise, it should return "false." Network Confessional meets this condition when nodes i and j are expected to sample the same packets (modulo loss and reordering). However, if node i is expected to sample more packets than node j (e.g., because it uses a larger sampling threshold), then node i is free to lie about the packets that should be sampled by i but not by j. This means that the receipt collector can verify X's performance, only based on the packets that are expected to be commonly sampled by X and its neighbors.

4.4 Adversarial Conditions

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

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 underperforming and misbehaving, or blames the losses on N and is exposed to N as a liar.

Delayed Forwarding. A dishonest domain may store every single packet, wait to learn whether the packet has to be sampled, *then* decide how to treat the packet. Such behavior can be beneficial, when the domain has multiple internal paths whose latencies vary by several milliseconds. We cannot prevent this attack, but we can render it increasingly expensive (hence impractical), by increasing the size of the circular buffer β: to realize the attack, a domain needs to equip its border routers with the capability to (1) temporarily store the β packets whose tuples are stored in the buffer and (2) index/forward these packets at line rate as the corresponding marker packets arrive. These capabilities require a significant amount of fast memory, which depends on the size of the buffer β, as well as custom router hardware, i.e., a domain would have to pay a router manufacturer to equip the domain's border routers with hardware, especially designed for nefarious goals. We discuss this further in Section 6.

Packet Crafting. Network Confessional was not designed to resist attacks where domains deliberately inject or modify packets (Section 2), 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.

5. PRACTICAL CONSIDERATIONS

We now relax the assumptions made in Section 4—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 [9]. But if NTP is not deemed sufficiently reliable, a domain can equip its border routers with radio or GPS receivers [2], 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 classifies 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 [8]. 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.

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 4.1, 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. First, we outline one possible implementation of Network Confessional that requires one 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 microseconds, independently of the table's size.

Each node uses a TCAM chip to store a $\langle Valid,\ PathID,\ PacketID,\ Time\rangle$ tuple for the β most recently observed packets. $Valid$ is a 1-bit field, and $PathID$ consists of the packet's source and destination prefixes. Upon receiving a new packet p, the node runs Alg. 1: if p is chosen as a marker, the node uses the TCAM search capabilities to identify all the stored tuples T with $T.Valid = 1$ and $T.PathID = T_p.PathID$, and sets $T.Valid$ to 0; from all the matching tuples, the node identifies the ones that satisfy $Hash\,(T.PacketID, T_p.PacketID) < \sigma$, and copies them elsewhere for later dissemination. New tuples are added in a circular manner, i.e., once the last entry of the table is reached, the node starts writing at the first entry, overwriting old (valid and invalid) tuples as it goes. Note that the order in which packets are stored in the buffer does not play any role—we can derive the order in which two packets were observed based on the timestamps of the corresponding tuples.

The feasibility and cost of this implementation are determined by the buffer size β, which must be on the order of $100\,000$, so that launching the delayed-forwarding attack (Section 4.4) is impractically expensive. Assuming this buffer size, the above implementation requires a TCAM chip of 2 MB with the ability to store 16 bytes (one tuple) per entry, which is readily available today. If a node wants to perform delayed forwarding, it needs to store the $100\,000$ most recently observed packets and forward each packet after it has become known whether this packet should be sampled or not; assuming 500 bytes per packet, this requires 50 MB of additional SRAM memory (to store the packets) plus additional custom functionality to index and forward the packets at line rate as the corresponding marker packets arrive. An ISP would need to order router hardware specially designed and configured for this purpose, in order to launch this attack.

We should note that there exist other implementations that store the tuples in SRAM and use TCAM only for the marker packets. These are less expensive in terms of required memory, but somewhat more complex conceptually, hence we chose not to discuss them here.

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) [3]. To be conservative, we consider a 10-domain path, where each border router samples 1% of the path's packets. Given receipt size (22 bytes), this path will incur an overhead of 0.2 bytes per packet; assuming 400 bytes per packet, this leads to a 0.046% bandwidth overhead for the path.

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 25 Gbps). This is not surprising, given that, *when fed realistic traffic*, a Nehalem server is bottlenecked at the I/O [10], 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 [20]. 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. PERFORMANCE EVALUATION

The key properties of Network Confessional that differentiate it from previous work are its bias-resistance, which is guaranteed by the fact that domains cannot guess future traffic, and tunability, which is guaranteed by the fact that domains mostly sample non-overlapping packet sets (Section 4). The key challenge was designing an implementation that did not require per-packet, per-flow, or per-path state, and showing that it can be realized with existing hardware and reasonable overhead (Section 6). Beyond these properties, Network Confessional is a random sampling mechanism: it measures the loss and delay incurred by a random subset of each domain's traffic and estimates the loss and delay incurred by the rest. Hence, the results we show in this section—that Network Confessional accurately estimates domain performance—will not come as a surprise. But that is precisely the point of this work: once we can efficiently implement bias-resistant, tunable sampling, measuring domain performance is straightforward, and we already know how to do it accurately.

We would like to explain why we do not compare Network Confessional to network tomography. One could argue that tomography is a natural candidate for comparison, since it enables end systems to estimate ISP performance. However, tomography was *designed* to work in scenarios where network nodes treat observed traffic independently from which path it belongs to and conditions in subsequent network links are uncorrelated. Network Confessional does not need these assumptions (this is why, unlike tomography, it requires support from the network). Hence, we could easily show simulation results where Network Confessional outperforms tomography by considering scenarios where domains selectively drop, delay, or throttle traffic from certain paths and/or the failures/congestion of subsequent links are correlated. We did not deem this to be either fair (to tomography) or interesting.

Methodology. Suppose we use Network Confessional to estimate the loss and delay performance of domain X from Fig. 1. Our goal is to compare these estimates to X's *actual* loss and delay performance under different traffic scenarios. Each experiment we present consisted of the following steps: (1) we took an actual packet trace and assumed that it corresponds to the traffic observed at node 4; (2) created a second, modified trace, by introducing loss, delay, and reordering in the first one, and assumed that it corresponds to the traffic observed at node 5; (3) generated the receipts that nodes 4 and 5 would generate if they observed the respective traffic; (4) estimated X's performance from its receipts as described in Section 4.3 and compared the estimates to X's actual performance.

For step 1, we used packet traces provided by CAIDA, collected in 2008 and 2009 from a Tier-1 ISP. For step 3 (i.e., to generate the receipts), we used our Click implementation (Section 6). For step 2, we introduced loss by discarding a subset of the packets from the original trace, assuming either uniform loss or the Gilbert-Elliot loss model [1] (the results were the same). Introducing delay was more complicated, as we are not aware of any commonly acceptable delay model for Internet traffic. Instead, we used the NS2 simulator to create two realistic congestion scenarios, and generated the sequence of delay values that a packet stream would encounter in each case. In both scenarios, nodes 4 and 5 were connected through a congested 10 Gbps intradomain path with a minimum latency of 50 msec. In the first scenario, our packet stream competed with an aggressive UDP flow that saturated the path; in the second one, it competed with hundreds of long-lived TCP flows.

Moreover, we wanted to introduce a certain amount of packet reordering. For this, we looked at the latest published reordering experiment that we could find, in which the authors sent packet streams of different rates along different Internet paths and measured the amount and type of reordering that occurred along each path as a function of the packet rate [12]. From the reported results, we chose those corresponding to the packet stream that incurred the largest amount of reordering—the one labeled F600(UDP, DC→LA, 1500) in [12]—to generate a reordering profile. In our experiments, this reordering profile caused about 0.5% of the packets that would have been sampled at some node to be reordered with a marker and not be sampled. The results reported in this section were derived by using this profile.

Before delving into performance evaluation, we performed a simple sanity check: we ran our implementation of Alg. 1 over several packet traces for different values of β, μ and σ, and verified that the resulting sampling rate for any given path follows Eq. 1. Once that was verified, we fixed β to 100 000 and $\frac{\mu}{\mathcal{M}}$ to 0.01 (i.e., chose 1% of the packets as markers) and varied only $\frac{\sigma}{\mathcal{S}}$ to change the sampling rate.

Loss Performance. Suppose that, for some period of time, domain X introduces loss rate λ into a given path. The question is, how accurately can we estimate λ? The

Loss (%)	# Samples		Time (seconds)	
	theoretical	actual	theoretical	actual
1	19 850	13 200	66.17	44.00
5	2300	2000	7.67	6.67
10	1000	815	3.34	2.72
15	600	760	2.00	2.54
25	315	330	1.05	1.10

Table 2: Samples and time needed to estimate loss with a relative standard deviation of 0.1.

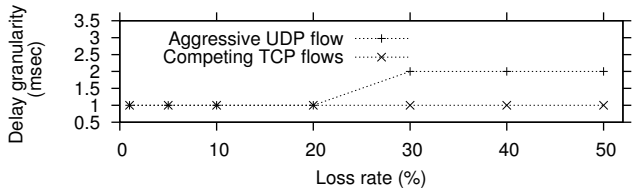

Figure 2: Granularity with which we can estimate the 90th quantile with confidence 90%.

answer depends on the number of samples used for the estimate, as well as the loss rate itself (Section 4.3). For instance, suppose that we want to estimate λ with a relative standard deviation of 0.1. Eq. 2 tells us that, if $\lambda = 1\%$ and $\pi_r = 0.5\%$, then we should compute our estimate after collecting receipts for $19\,850$ packets; however, if $\lambda = 25\%$ (and π_r is the same), then we only need receipts for 315 packets. To see what happens in practice, we looked at different traces and different paths from them, introduced different amounts of loss into each path, and estimated this loss as a receipt collector would.

Table 2 shows results for a representative path. The first column specifies the loss rate λ. The second column shows how many samples we need according to Eq. 2, in order to estimate this loss rate with a relative standard deviation of 0.1. The third column shows how many samples we needed in practice in order to achieve this accuracy (e.g., for $\lambda = 1\%$, we had to collect $13\,200$ samples in order to reach a relative standard deviation of 0.1). The fourth and fifth columns show how much time it would take to collect the respective number of samples given a path of $30\,000$ packets/sec (roughly 100 Mbps, assuming 400 bytes/packet) and a sampling rate of 1%. Consistently with the theory, the lower the loss rate that we tried to estimate, the larger the number of samples that we needed in order to estimate it accurately. However, even a loss rate of 1% was accurately estimated in less than 1 minute.

Delay Performance. Now suppose that we are trying to estimate X's delay distribution with respect to a given path, over a certain period of time. As described in Section 4.3, we use the technique from [22], which gives us an upper and a lower bound for the delay distribution, which hold with probability π. We ask two questions: how far are these upper and lower bounds from each other in different congestion scenarios, i.e., with what *granularity* can we estimate the delay distribution? and how does this granularity change in the face of loss?

Again, the answer to the first question depends on the number of samples used for the estimation. We computed the upper and lower bounds for different paths and our aggressive-UDP-flow and competing-TCP-flows congestion scenarios, after collecting receipts for 1200 packets from each path (given a path of $30\,000$ packets/sec and a sampling rate of 1%, 1200 sampled packets correspond roughly to $T = 4$ sec). In all cases, the bounds stayed within one or two milliseconds from one another. Of course, the credit for this result goes to the authors of [22], who devised the technique for obtaining the bounds. Still, we mention this result, to make the point that their technique matches well our goal of accurately estimating the delay performance of network domains in the presence of severe congestion (which is when they would mostly want to hide it).

The second question is more relevant to our contribution. Intuitively, the more loss we have between nodes 4 and 5, the fewer the packets that are commonly sampled by the two nodes over the given time period, hence, the worse the granularity with which we can estimate X's delay distribution. Fig. 2 focuses on the 90th delay quantile and shows how the granularity with which we can estimate that quantile with probability $\pi = 90\%$ changes as a function of loss. We see that, in the aggressive-UDP-flow scenario, loss rate does affect granularity, but not to a practically significant extent—introducing 50% loss decreases granularity barely by 1 msec.

8. 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. [23] describe a taxonomy of identification schemes for network adversaries who drop packets. Delayed disclosure comes in the form of an explicit request from the source identifying the packet that should be acknowledged, to identify who dropped it. That work targets a stronger adversarial model (e.g., adversaries who may modify or inject packets) but relies on stronger assumptions as well: explicit signaling in addition to normal traffic, symmetric traffic paths, application-layer processing of all receipts by all nodes on a path (onion cryptography). In contrast, Network Confessional requires no explicit signaling using instead later traffic to derive late-disclosed secrets, makes no claim about the path traversed by receipts to collectors, requires processing only by the issuer of a receipt. 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 [6] and the fault-localization protocols from [13] inform traffic sources where individual packets get lost or corrupted. AudIt provides source domains with similar *per-TCP-flow* information [7]. 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 [11]. 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 [17]. The "Secure Sketch" technique from [14] 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 3, 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 [18].

9. CONCLUSIONS

We have presented Network Confessional, a system by which network domains can estimate and verify each other's loss and delay performance. Network Confessional relies on domains producing and exchanging receipts for the traffic they receive and deliver. A domain can estimate a neighbor's performance by processing the receipts produced by the neighbor; it can verify that the neighbor's receipts are honest by comparing them to the receipts produced by other domains for the same traffic. If a domain lies about its performance, that leads to receipt inconsistencies and exposes the liar to its neighbors. Network Confessional comes at the cost of deploying (modest) new functionality at domain boundaries. The processing, memory, and bandwidth overhead incurred by a deploying domain is configurable and independently determined by the domain.

Acknowledgments. We would like to thank Marco Canini, Olivier Crameri, Mihai Dobrescu, Gianluca Iannaccone, Ming Iu, Sylvia Ratnasamy, Vyas Sekar, Nina Taft, and our shepherd, John Byers, for their invaluable help.

10. REFERENCES

[1] Gilbert-Elliot Loss Model. `http://www.eecs.tu-berlin.de/fileadmin/fg112/Papers/tkn_report02.pdf`.

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

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

[4] Ofcom Reveals UK Real Broadband Speeds. `http://www.ofcom.org.uk/media/features/broadbandspeedsjy`, 2009.

[5] Office of Communications, Traffic Management and Net Neutrality. `http://www.ofcom.org.uk/consult/condocs/net-neutrality/summary/`, June 2010.

[6] 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.

[7] 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.

[8] K. Argyraki, P. Maniatis, and A. Singla. Verifiable Network-Performance Measurements. Technical report, EPFL, Switzerland, November 2010.

[9] 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.

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. van Wesep, T. Anderson, and A. Krishnamurthy. Reverse Traceroute. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2010.

[16] E. Katz-Bassett, H. V. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson. Studying Black Holes in the Internet with Hubble. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, April 2008.

[17] 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.

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

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turn. OpenFlow: Enabling Innovation in Campus Networks. *ACM Computer Communications Review*, 38(2), April 2008.

[20] 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.

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

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

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