Measuring Path Quality in the Presence of Adversaries: The Role of Cryptography in Network Accountability

Sharon Goldberg, David Xiao, Boaz Barak, and Jennifer Rexford
Princeton University

ABSTRACT

Mechanisms for measuring data-path quality and identifying locations where packets were dropped are crucial for informing routing decisions and enforcing network accountability. If such mechanisms are to be reliable, they must be designed to prevent ASes from 'gaming' measurements to their advantage (*e.g.,* by hiding packet loss or by blaming packet loss on innocent ASes). In this paper, we explore mechanisms for accurately *detecting* and *localizing* packet loss events on a data path in the presence of both benign loss (congestion, link failure) and active adversaries (ASes motivated by malice or greed). We do not advocate a specific network architecture. Instead, we use rigorous techniques from theoretical cryptography to present new protocols and negative results that can guide the placement of measurement and security mechanisms in future networks.

Our major contributions are: (1) Negative results that prove that any detection or localization mechanism requires secret keys, cryptography and storage at every participating node. (2) *Pepper Probing* and *Salt Probing*, two efficient protocols for accurate end-to-end detection of packet loss on a path, even in the presence of adversaries. (3) A new protocol for accurately localizing packet loss to specific links along a path, even in the presence of adversaries.

1. INTRODUCTION

The Internet's best-effort service model provides neither *a priori* guarantees of packet delivery nor *post hoc* information about the fate of dropped packets. However, tools for measuring data-path quality and identifying locations where packets were dropped are crucial for informing routing decisions at the edge of the network (*e.g.,* in source routing, intelligent route control, and multipath routing). Moreover, [16] recently argued that the Internet industry's resistance to evolution *cannot* be overcome without an accountability framework that measures path quality and then holds ISPs responsible for poor path performance. The combination of an accountability framework and tools for intelligent routing could counter the growing trend of commoditization on the Internet by giving ISPs an economic incentive to upgrade their networks in order to attract customers [2, 8, 16].

In this paper we explore methods that empower the edge of the network to measure data path quality. Because the network layer is typically responsible for making routing decisions, we focus specifically on tools that source edge networks (*i.e.,* stub ASes or edge routers) can use to perform the following two types of measurements: *fault detection (FD)* to detect faults on a data path, and *fault localization (FL)* to localize the links at which the faults occurred. We say that a fault occurs when a packet sent by a source edge network fails to arrive unaltered at its destination edge network within a reasonable amount of time.

The presence of adversaries: The design of robust measurement schemes is complicated by the presence of parties on the network that have a *strong incentive to bias path quality measurements.* Consider these natural scenarios:

- ISPs may provide poor service to selected classes of traffic in order to cut costs. ISPs have an economic incentive to prevent their customers from taking their business elsewhere or from demanding accountability for service level agreement (SLA) violations. As such, greedy ISPs may attempt to bias measurements to prevent edge networks from detecting degraded path quality, or even to blame degraded service on an innocent ISP.
- A malicious router or AS, corrupted by a remote attacker or disgruntled network operator, may advertise routes through himself so that he can selectively block or tamper with certain flows. For example, a remote attacker may tamper with packets containing information about a corporation's stock price, sent by a website like CNN.com. The hacked router has a strong incentive to bias measurements so that his malicious behavior can continue undetected, or is attributed to an innocent router.
- A router may 'benignly' drop packets due to malfunction, misconfiguration or excessive congestion. For example, an MTU (Maximum Transmission

Unit) mismatch along the path might cause a router to drop large packets, while continuing to forward small packets (*e.g.,* from ping and traceroute). Rather than making assumptions about the nature of packet loss events, we can instead assume that packet loss occurs in the "worst possible way", *i.e.,* according to a pattern that introduces bias in our measurements.

As such, we avoid making *ad hoc* assumptions about the good behaviour of ASes or routers on the data path. Instead, *we assume that a source edge-network can trust only the destination edge-network at the end of the data path she is measuring, while the intermediate nodes on the path may adversarially affect her measurements.* We focus on finding efficient protocols and minimal resource requirements for fault detection and localization; to avoid burdening our schemes with extra machinery, our model does not consider traditional security issues like preventing faults, providing confidentiality, or protecting against traffic analysis or denial-of-service attacks.

Accountability, Security and Measurement: Our work lies at the intersection of the literature on network accountability, data-plane security, and path-quality measurement. The literature on accountability [2, 16] focuses on ASes' economic motivations for making decisions, but it does not explicitly consider mechanisms to prevent ASes from 'gaming' an accountability framework to their economic advantage. On the other hand, much of the work [3,5,12,18,21,22] on data-plane security implicitly assumes that faults caused by adversarial nodes should be detected on a *per-packet* basis (*i.e.,* when just one packet is dropped [3, 5, 12, 22]). Here we consider benign faults (*i.e.,* congestion, link or node failures) alongside adversarial behaviour, which makes our models simultaneously more realistic and more difficult to analyze. Moreover, in addition to considering the per-packet approach, we also explore *statistical* approaches for estimating *average fault rate* (*i.e.,* the fraction of packets that experienced faults) on the data path. Finally, our approach can be seen as a secure analogue of the statistical path-quality measurement techniques designed by the Internet measurement community [9, 10, 13, 17, 24] for the adversarial setting. Our work is most closely related to that of Stealth Probing [4], a statistical fault detection protocol designed for a similar adversarial setting.

Techniques from theoretical cryptography: In contrast to previous work, *in this paper we use a rigorous theoretical framework to explore a wider space of solutions for the adversarial setting; we consider both per-packet and statistical approaches for performing fault detection (FD) and fault localization (FL).* In fact, by working within the framework of theoretical cryptography to precisely define security for our adversarial setting, we have exposed security vulnerabilities of other FD and FL protocols [2, 5, 10, 21, 25] (we discuss these in the body and footnotes of the paper). Furthermore, our rigorous approach has allowed us to *prove* the security of our FD and FL measurement protocols, and to use the techniques of [14, 15, 19] to present a set of *negative results* that determine the minimum resources necessary to build any FD or FL protocol that is robust to adversaries. Though our security definitions consider active adversaries with extensive powers (Section 2, 3.1 and 4.1), many of our negative results hold even in a weaker adversarial setting (see Sections 3.2 and 4.2).

Cast of characters: We begin by introducing the reader to Alice, a source edge network, (edge router or stub AS), who sends data packets to Bob, a destination edge network. Eve, the adversary, occupies some subset of intermediate nodes on the path. *In this paper, a node can be either an AS or an individual router.*

Results on fault detection (FD): We present negative results (Section 3.2) that prove that *any* secure per-packet or statistical FD protocol requires (1) a key infrastructure, (2) cryptographic operations at Alice/Bob, and (3) dedicated storage at Alice. Our results imply that any scheme that does not make use of these resources *cannot* be secure in the adversarial setting. Next, we present a simple secure per-packet FD protocol, *Per-Packet Ack*, and two novel secure statistical FD protocols, **Pepper Probing** and **Salt Probing**. In Per-Packet Ack, Bob acknowledges each packet that Alice sends with an authenticated ack packet (Section 3.3). Pepper Probing and Salt Probing are efficient protocols that allow Alice to estimate the average fault rate, up to an arbitrary level of accuracy, without requiring any modifications to Alice's traffic. In Pepper Probing (Section 3.5), Alice and Bob sample packets in a coordinated manner with a cryptographic hash function, similar to the approach of Trajectory Sampling [10]. Salt Probing (Section 3.6) extends Pepper Probing to the public-key setting via a timing strategy similar to that of TESLA [23].

Results on fault localization (FL): We present a set of negative results that prove that *any* secure per-packet FL protocol requires all the resources required for secure per-packet FD and in addition, each intermediate node must (1) share keys with Alice, (2) perform cryptographic operations, and (3) keep dedicated storage (Section 4.2). Next, we limit ourselves to the setting of source routing with symmetric paths and present an **Optimistic Protocol** for per-packet FL (Section 4.3), and a **composition** technique for constructing a secure statistical FL protocol by having each intermediate node simultaneously run Pepper or Salt Probing to the destination edge network (Section 4.5). Our FL protocols can guide future designs of FL protocols outside the source-routing setting.

Technical highlights: This paper is a rigorous theoretical exploration of solution space for FD and FL; rather than advocating a specific network architecture, we present new protocols and negative results that can be used to guide the placement of measurement and security functions in future networks (Section 5). Our protocols leverage a variety of useful cryptographic techniques, including: (1) using randomly-selected *salt* values to run symmetric cryptographic operations in the public-key setting (Section 3.6), (2) using *onion reports* to prevent an adversary from blaming bad behaviour on an innocent node (Section 4.3), and (3) using a *composition* technique to construct a statistical FL protocol out of statistical FD protocols (Section 4.5). Moreover, while some of our proofs follow quite simply from our security definitions, many of our proofs are more involved (*e.g.,* the necessity of cryptography for FD and FL). *Because we choose to present the entire breadth of our results, in this paper we provide only proof sketches and defer full proofs to our technical report [1].*

2. OUR SETTING

Goals and non-goals: When considering path quality we focus on both *availability* (path delivers packets to the correct destination) and *integrity* (packets arrive unaltered). Therefore, we say that a fault occurs when a packet sent by Alice fails to arrive unmodified at Bob within a reasonable amount of time. Our techniques can be extended to obtain finer measurements, such as calculating the average round-trip time (RTT); however, to simplify the presentation we do not discuss RTT measurements here. In this paper, we do *not* study techniques to *prevent* an adversary from adding packets to the data path or from (selectively or fully) causing faults, nor do we consider techniques that guarantee confidentiality or protect against traffic-analysis attacks. Moreover, our protocols are *not* designed to *diagnose* the cause of a fault. In the adversarial setting it is extremely difficult to accurately distinguish between benign and adversarial behaviour because an adversary can always drop packets in a way that 'looks like congestion'. As such, we do not distinguish between faults caused by benign causes (*e.g.,* congestion, node failure), malicious behaviour, or even increased congestion on a link during a denial-of-service (DoS) attack. Finally, for lack of space, we do *not* explicitly address the problem of *protecting* our FD and FL protocols themselves from DoS attacks (*e.g.,* an adversary crashes an FD monitor by flooding it with messages that require verification of a digital signature); however, standard rate-limiting techniques can mitigate these attacks.

Adversary model: We consider both benign faults (due to congestion or node failure) and malicious faults (caused by an active adversary). We assume that Alice cannot trust any intermediate node along the data path, except for Bob. One (or more) nodes on the data path may be controlled by an active adversary, Eve, with extensive powers: she may add, drop, or modify traffic on the data path; she may observe the traffic on the data path and use any observations, including timing information, to learn how to bias Alice's measurements; she may attempt to blame her own malicious behaviour on innocent nodes, *e.g.,* by simulating the behavior of the system during past instances of congestion-related loss, as in Section 3.2; she may share information with other adversarial nodes on the data path or collude with those nodes (*e.g.,* by tunnelling packets).

Per-packet vs. statistical approaches: In per-packet schemes, Alice learns if a fault occurred for *each packet* that she sends to Bob. To reduce the overhead of per-packet schemes, we also consider statistical schemes, where Alice instead estimates the average fault rate on the data path. We believe that statistical schemes are sufficient for most situations; because it is natural that some packets are dropped due to congestion, we argue that the network layer neither cares nor expects to know the fate of every single packet sent. On the other hand, if the network layer wants to detect small transient problems which selectively harm specific end-hosts (*i.e.,* drops of one or two TCP SYN packets), then per-packet approaches are more appropriate.

Resources and Evaluation Criteria: We shall evaluate our protocols based on how efficiently they use the following five resources: We prefer protocols that minimize (1) *communication overhead*, (2) *computation* of cryptographic operations, and (3) use of dedicated *storage* in the router. Furthermore, the difficulty of building up and maintaining a (4) *key infrastructure* for the Internet is well known. As such, we prefer schemes that require few keys; public-key schemes, where each node has only a single key, are preferred over schemes where each pair of nodes share a symmetric secret key. Finally, we prefer protocols that (5) *do not affect the router's internal packet-processing path.* In fact, all our protocols do not modify any packets sent by the source edge-network, so that they can implemented in a monitor located off the critical path in the router. This approach has the additional advantage of minimizing latency in the router, not increasing packet size, and allowing Alice to turn measurements on and off without having to coordinate with Bob.

3. FAULT DETECTION

Secure fault detection (FD) is an end-to-end technique that allows Alice to detect whether or not her traffic arrived unaltered at Bob, in the presence of adversaries on the data path. In this section, we start by defining security for per-packet FD, present a set of negative results that shows the resources required for any secure FD protocol, and then present a simple secure per-packet FD protocol. We then define security

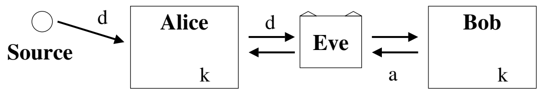

Figure 1: FD security game.

for statistical FD and present two novel statistical FD protocols, Pepper Probing and Salt Probing.

We focus on FD protocols with the simple structure shown in Figure 1: for each data packet d generated by an end-host, Alice optionally stores information about d in memory and then sends d to Bob. Bob optionally replies with an ack $a = \mathsf{Ack}_k(d)$. (In practice, many acks a may be batched together and sent in a single packet.) We say an *exchange* is all the communication associated with transmitting a single data packet d.

3.1 Per-packet FD: Security Definition

In theoretical cryptography, game-based definitions are used to obtain precise guarantees of security [11]. We now present a game-based definition for secure per-packet FD, consisting of a description of the game setting, followed by correctness and security conditions.

Definition 3.1 (Secure per-packet FD)**.** The game setting for per-packet FD, shown in Figure 1, defines the power of the adversary (Eve) and models how she interacts with the honest parties (Alice, Bob). Because Eve is a node on the data path between Alice and Bob, we assume that all communication between Alice and Bob is controlled by Eve. Eve can do whatever she wants to the packets she receives from Alice and Bob; of course, if she was behaving honestly, she would forward packets along unmodified. We use the 'Source' entity in Figure 1 to model the end-hosts that generate the data packets d that Alice sends Bob. We assume that the data packets generated by Source are unique.[1] The game consists of Source repeatedly sending packets and Eve playing with the packets she gets from Alice and Bob until Eve decides to stop the game.

Correctness condition: Here we define what should happen when there is no adversarial behaviour We say that an *FD protocol is correct* if when Alice and Bob interact, Bob always receives the packet d sent by Alice, and Alice never detects a fault.

Security condition: Here we define what the adversary needs to do in order to break the security of an FD protocol. We say that *Eve wins the per-packet FD game* if, during the course of the game, she manages to drop or modify *a single packet* while still convincing Alice that no fault occurred (*i.e.,* because Alice saw a valid ack for each packet that she sent).

A per-packet FD scheme is secure if no efficient adversary Eve is able to win the per-packet FD game with non-negligible probability.

Monotonicity: Our security definition is *monotone* in that it protects not only against adversaries like Eve in Figure 1, but also against adversaries like Helen (a node *off the data path monitored by Alice* who attempts to trick Alice into detecting faults on the path to Bob so that Alice will switch her traffic to a path through Helen). We assume that Helen can only *add* packets to the data path between Alice and Bob. Observe that Helen could potentially trick Alice into detecting a fault (when no fault occurred) by sending Alice invalid ack packets spoofed to look like they came from Bob. However, our definition protects against attacks by Helen because the definition is *monotone*: as long as Alice receives a valid ack for every packet she sends, she will never declare a fault, *even if she receives additional nonsense packets.*

Congestion: Our definition does not explicitly model congestion. FD is an end-to-end measurement; as such, it is sufficient to detect the total fault rate over an entire path, without distinguishing between faults caused by normal congestion or adversarial behaviour.

Faulty forward or reverse path? Collecting one-way measurements of a path is notoriously difficult, since a sender cannot easily differentiate between faults on the forward path (from Alice to Bob) and faults on the reverse path (from Bob to Alice). As such, we define our FD protocols so that Alice always obtains a conservative estimate, *i.e.,* a lower bound, on the fault rate on her forward path. This definition works best in the setting of *symmetric paths*, where the forward and reverse paths are identical, since here Eve has no incentive to drop the acks on the reverse path. (If she does, she simply makes the path look worse). In contrast, in the setting of *asymmetric paths*, it is useful to distinguish between faults that occur on the forward path and faults that occur on the reverse path. In this setting, Eve occupying only the reverse path may have an incentive to drop acks, perhaps to confuse Alice into thinking that the forward path is faulty.

We argue that any FD protocol that distinguishes between faults on the forward vs the reverse path must also include a coordinated path-switching mechanism between Alice and Bob, even in the benign setting. To see why, observe that Alice cannot distinguish between (a) the forward path failing and (b) the reverse path failing. In both situations, Alice cannot communicate with Bob. However, to learn why communication failed, in case (a) the forward path must be switched, while in

[1]We make this assumption to prevent Eve from trivially winning the FD game using a *replay attack*; suppose Source sends the same data packet d twice. The first time, Eve forwards d to Bob and gets valid ack a. The second time she *drops* d but responds to Alice with the valid ack a. In practice, we can prevent replay attacks by timestamping Bob's acks with an expiry time, such that no repeated packets are sent, (*e.g.,* due to natural entropy in packet contents, TCP sequence numbers, and IP ID fields) for the duration of the time interval for which Bob's acks are valid.

case (b) the reverse path must be switched. Because path-switching mechanisms are outside our scope, in this paper we do *not* construct FD protocols that explicitly distinguish between faults on the forward vs the reverse path. While we leave this interesting problem to future work, we note that such FD protocols are still subject to our negative results, and they could be designed using our simpler (Per-Packet Ack, Pepper and Salt Probing) FD protocols as building blocks.

3.2 Negative Results

Here we show that *any* FD scheme that is secure according to the per-packet definition in Sections 3.1 (1) requires shared keys between Alice and Bob, (2) requires Alice/Bob to perform some cryptographic operations,[2] and (3) requires Alice to modify her memory for each packet that she monitors. All these results hold even in the statistical FD model of Section 3.4, and even if the FD protocol is allowed to modify Alice's traffic or has a more general structure than the simple two-message structure of Figure 1. Furthermore, while our results about the necessity of keys and cryptography rely on the assumption that Eve actively forges acks, our result about the necessity of storage holds even when Eve's adversarial powers are limited to dropping packets. Our negative results imply that the resource overhead of our Per-Packet-Ack, Pepper and Salt Probing protocols (Sections 3.3, 3.5 and 3.6) are unavoidable in the sense that we could not have designed them without keys, cryptography and storage.

Keys are necessary: We prove this in the contrapositive. Assume that Alice and Bob have no shared secret key. It follows that Eve knows everything that Bob knows, so that Eve can construct Bob's acks on her own. Therefore, Eve could then drop all the packets going to Bob, while convincing Alice that nothing is wrong, and the FD scheme cannot be secure.

Cryptography is necessary: We now prove that the keys must be used in a 'cryptographically-strong' manner by showing that any secure FD protocol is at least as complex as a secure **keyed identification scheme (KIS)**. In KIS, Alice and Bob share a secret key, and Alice wants to ascertain Bob's identity. She achieves this by asking him to solve a riddle that can only be solved by someone who knows the secret key. A KIS is secure if an impersonator who does not know the key can't solve the riddle with better success than by just guessing a random answer. To prove that any secure FD scheme is at least as complex as a secure KIS, we show that any secure FD scheme can be used to construct a secure KIS. The construction is simple: the riddle in our new KIS is a randomly chosen packet d

[2]Listen [25] is an FD protocol that does not require Alice or Bob to perform cryptographic operations. Our results imply that Listen is insecure in our adversarial setting.

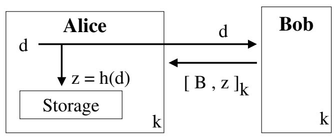

Figure 2: Per-Packet Ack protocol.

(in the FD scheme). The correct answer to the riddle in our new KIS is the ack $a = \mathsf{Ack}_k(d)$ (from the FD scheme). We complete the proof in [1] with a *reduction* that proves that if the FD scheme used in the above construction is secure, then our new KIS construction is also secure. *Our proof implies that secure identification is an unavoidable part of secure FD.* That is, any secure FD protocol must ensure that Alice can distinguish between acks sent by Bob, and acks sent by an adversary impersonating Bob. Furthermore because KIS is known to be at least as complex as symmetric-key encryption [14, 19], our proof implies that *one cannot expect to design secure FD protocols that run blazingly faster than symmetric-key encryption protocols.*

Alice must modify memory for *each monitored packet*: This is easily proved in the contrapositive. Assume that Alice stores no information (*i.e.,* does not modify her memory) about a packet d for which she is expecting an ack. It follows that Eve can simply drop d after Alice sends it, and Alice won't notice (because Alice immediately *forgets* about d after she sends it), and the FD scheme cannot be secure.

3.3 Per-Packet-Ack: A Simple FD Protocol

Per-Packet Ack is a simple secure per-packet FD protocol, where Bob securely acknowledges receipt of every packet he sees. Despite the simplicity of the protocol, shown in Figure 2, we explain it here in some detail in order to introduce the reader to some of the cryptography we use in later parts of the paper.

The protocol is built from two cryptographic primitives: A **collision-resistant hash function (CRH)** [11], which we denote by h, is a hash function for which it is computationally infeasible to find two inputs $x \neq x'$ that map to the same output $h(x) = h(x')$. In practice h can be implemented with a function such as SHA-2 [20]. A secure deterministic **message authentication code** (MAC) [11] protects a message's integrity and authenticity by allowing a receiver, who shares a secret key k with the sender, to detect any changes made to the sender's message. A MAC is secure if no efficient adversary can forge a valid signature on any arbitrary message (with non-negligible probability). We use the notation $[m]_k$ to denote message m MAC'd with key k.

Details of protocol: In the Per-Packet-Ack, Alice and Bob share a symmetric secret key k. For each packet d that Alice sends, she stores a packet digest[3]

[3]Packet digests must be computed only on *immutable* fields

p:	Fraction of packets acknowledged by Bob.
α:	False alarm threshold. Alice avoids raising an alarm when the fault rate is below α.
β:	Detection threshold. Alice raises an alarm when the fault rate exceed β.

Table 1: Parameters for statistical FD.

$z = h(d)$ along with a time-out (we say that a fault occurs if the time-out expires before Alice receives an ack for that packet). When Bob receives the packet d, he computes the packet digest $z = h(d)$ and sends an ack of the form $a = [B, z]_k$ where B is a public unique identifier for Bob's identity. Alice removes a packet digest z from storage when she receives an acknowledgment a containing a matching packet digest z and a valid MAC. Periodically, Alice checks her storage to see if any packets awaiting acknowledgment have timed out; if so she raises an alarm and declares that a fault occurred.

Security: Notice that in this protocol, acks are *unambiguous*:[4] they specifically depend on the contents of the packet d that Bob receives. It follows that if Eve wants to create a valid ack to a packet d that was dropped before it reached Bob, she either has (i) to find a collision d' in the CRH h, so that $h(d) = h(d')$, and ask Bob to generate an ack for d', or (ii) she has to forge the MAC signature on an ack $(B, h(d))$ without Bob's help. By the security of the CRH and the MAC, both (i) and (ii) occur with negligible probability, so Per-Packet-Ack is secure.

3.4 Statistical FD: Security Definition

The per-packet FD protocol of Section 3.3 required acks and memory modifications every packet sent. To reduce the high overhead required for per-packet protocols, we now study techniques that require only an accurate measurement of the average fault rate. We will focus on schemes that use sampling.

Sampling: Instead of acknowledging every packet, as in Per-Packet Ack, in sampling schemes we randomly select a p fraction of the packets to monitor in order to obtain an estimate of average behavior. Our estimate becomes accurate if we obtain a sufficient number of samples. We let *probing schemes* denote statistical FD schemes that use sampling, and *probes* denote packets that require acks, and $p < 1$ denote the fraction of packets sent by Alice that are acknowledged by Bob. Security in the statistical setting guarantees that *Eve cannot bias Alice's estimate of the average fault rate.*

of the packet, that are unchanged as the packet traverses the data path. See [10] for immutable fields in an IP packet.

[4]We note that because the security of the scheme relies on acks being unambiguous, a protocol in which Bob sends back a count of the number of packets received (*e.g.,* as suggested in one version of Fatih [18]) is not secure in our setting.

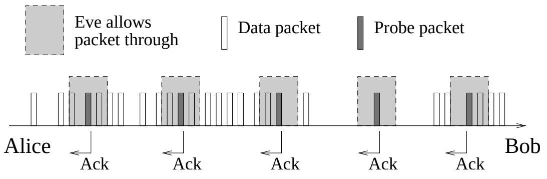

Figure 3: Attack on active measurement.

Probe Indistinguishability: Any probing scheme that is secure in the adversarial setting requires a property called probe indistinguishability [2, 4]. That is, to prevent Eve from biasing Alice's measurements by treating probe packets preferentially (while, say, dropping all other packets), *Eve must not be able to distinguish probe packets from non-probe packets.* In [1] we discuss why timing attacks make it difficult to ensure probe indistinguishability in FD protocols based on *active measurement* (*e.g.,* ping, traceroute, and the approaches of [13, 17, 24]). In active measurement, Alice injects new, specially-crafted probe packets into her traffic stream. To illustrate these timing attacks, consider an extreme example where probes are injected according to a periodic arrival process as in Figure 3. Eve can easily learn how to distinguish probes from existing traffic by observing when Bob sends ack packets. Eve can then drop all packets sent outside a small time window around the beginning of the probe period. As such, we avoid schemes based on active measurement, and instead focus on *passive sampling* protocols that sample from existing traffic.

Definition 3.2 ((α, β)-Statistical FD Security)**.** α denotes a fault rate that gives acceptable network performance, while β is the fault rate above which Alice decides that a path is unacceptable. The game setting is the same as Definition 3.1 with the two modifications: First, Source must send at *at least T packets.* Second, at the end of the game, Alice either raises an alarm if she decides that Eve caused the fault rate to exceed a *detection threshold* $\beta \in [0, 1]$, or Alice does not raise an alarm if she decides the fault rate was below a *false-alarm threshold* $\alpha \in [0, 1]$. [5]
Correctness condition: A *statistical FD protocol is correct* if, with probability greater than 99%, Alice does not raise an alarm when less than an α-fraction of exchanges contain faults (due to Eve or congestion).
Security condition: A *statistical FD protocol is secure* if, with probability greater than 99%, Alice raises an alarm when Eve causes faults for more than a β-fraction of exchanges.

[5]In this paper we discuss only (α, β)-security, where Alice *raises an alarm* when the fault rate exceeds a threshold. In [1] we use a more precise definition where Alice also gets a sufficiently accurate *estimate* of the fault rate on the path. Pepper and Salt Probing satisfy both definitions.

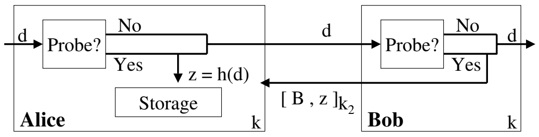

Figure 4: Pepper Probing.

Duration of the game: Notice that our definition explicitly puts a lower bound on T, number of exchanges for which the security game is played. This bound on T is *essential* because a statistical scheme only measures behaviour on average; Alice can only have high confidence in her measurements when her sample size is sufficiently large.

3.5 Pepper Probing: Statistical FD

Pepper Probing extends the Per-Packet Ack protocol of Section 3.3 to the statistical setting by allowing Alice and Bob to sample packets in a coordinated way by computing a 'cryptographic hash' over packet contents (similar to Trajectory Sampling [10]'s approach).

Details of protocol: As shown in Figure 4, Alice and Bob share a secret $k = (k_1, k_2)$. For each packet sent, they use k_1 to compute a function Probe that determines whether or not a packet d is a probe and should therefore be digested $z = h(d)$, stored, and acknowledged. To acknowledge a probe, Bob sends Alice an ack that is MAC'd using k_2 to key a secure MAC, as in Per-Packet Ack (Section 3.3). The Probe function is implemented using a **pseudo-random function (PRF)**[6]. A PRF [11] is a keyed function that cannot be distinguished by any computationally-efficient algorithm from a *truly random function.* (A truly random function maps each distinct input to a truly random output.) We can think of a PRF as a keyed hash function (keyed here with k_1) that takes in an input (here, the data packet d), and outputs a number $f_{k_1}(d) \in \{1, \ldots, 2^n\}$. We let

$$\begin{array}{ll} \mathsf{Probe}_k(x) = \text{Yes} & \text{if } \frac{f_{k_1}(d)}{2^n} < p \\ \mathsf{Probe}_k(x) = \text{No} & \text{otherwise} \end{array} \qquad (1)$$

At the end of the probing session, Alice computes an estimate F of the fault rate of the game by computing the fraction of the probe packets that experienced faults (*i.e.,* were not properly acknowledged by Bob). If $F > \frac{\alpha+\beta}{2}$ she raises an alarm, while if $F < \frac{\alpha+\beta}{2}$ she decides that everything was normal.

Security: First, observe that as in the Per-Packet Ack protocol (see Section 3.3), (i) Eve cannot forge a valid ack with non-negligible probability. Next, we argue that $\mathsf{Probe}(d)$ satisfies probe indistinguishability. Recall that we assume that all the data packets in the security game are distinct (Section 3.1) and that PRFs are indistinguishable from truly random functions. It follows that the Probe function will select an (approximately) random p-fraction of data packets as probes. It follows that (ii) no efficient adversary Eve who sees (or even chooses) the data can predict whether or not a packet is a probe with better probability than just guessing randomly with probability p. Now, (i) and (ii) imply that the probability that Alice detects each fault is very close to p. Using a Chernoff bound [11] (which tells us the estimated fault rate should be close to true fault rate if the number of exchanges sampled T is sufficiently large), we can show that (iii) when the actual fault rate is below the false-alarm threshold α, the probability that Alice's estimate of the fault rate F exceeds threshold $\frac{\alpha+\beta}{2}$ is at most $e^{\frac{-p(\beta-\alpha)^2 T}{12\alpha}}$, and (iv) when the actual fault rate is above the detection threshold β, the probability that Alice's estimate of the fault rate F is below $\frac{\alpha+\beta}{2}$ is also at most $e^{\frac{-p(\beta-\alpha)^2 T}{12\beta}}$. It follows from (iii), (iv) and Definition 3.2 that Pepper Probing is both correct and secure when the protocol is run on at least T packets where

$$T > \frac{64\beta}{p(\beta-\alpha)^2} \qquad (2)$$

Security fails without pseudo-randomness: Suppose that the Probe function of Equation 1 was implemented using a CRC keyed with a secret modulus, as in [10], instead of a PRF. Model the CRC function as $f_k(x) = x \mod k$, and consider the following attack: Eve starts by observing the interaction between Alice and Bob, and recording a list of packets that were not acknowledged by Bob. Then, whenever she sees a new packet that is within a small additive distance of old packet that was not acknowledged, she drops the packet. Thus, Eve can drop non-probe packets with high probability, and she can bias Alice's estimate F below the true fault rate. This attack is possible because the CRC does not use its 'secret key' in 'cryptographically-strong' manner (*c.f.,* our result on the necessity of cryptography, Section 3.2).

Efficiency: For a reasonable set of parameters, say overhead $p = 0.02$, false-alarm threshold $\alpha = 0.005$, and detection threshold $\beta = 0.01$, it follows from Equation 2 that Pepper Probing is (α, β)-secure when the protocol runs for at least $T > 1.3 \times 10^6$ packets. If each packet digest is 128 bits long, it follows that Alice needs about $pT \times 128 = 3.3$ Mbits of storage. (While Alice requires this much storage for each Bob network, she can choose to run Pepper Probing with only a small number of Bob networks at a time by ignoring all the acks sent by all other Bob networks.) Pepper Probing incurs only a small communication overhead of p (due only to to acks; Alice's traffic is unmodified). Consider now the Stealth Probing protocol of [4], another secure statistical FD protocol which also uses passive sampling and

[6]The high-throughput PRF we require here can be efficiently implemented in hardware with pipelined AES in CBC-mode or with a cryptographic hash function [20].

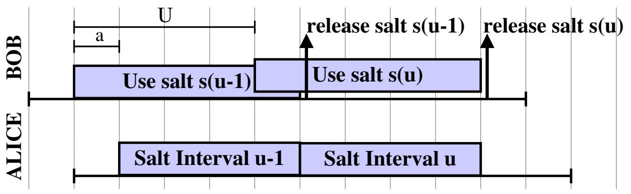

Figure 5: Timing for Salt Probing.

authenticated acks for a p-fraction of packets. Whereas Stealth Probing requires Alice to authenticate and encrypt all of her traffic, Pepper Probing does not require any modifications to Alice's traffic.

3.6 Salt Probing: Public-Key Statistical FD

While Pepper Probing requires pairwise symmetric keys between each Alice-Bob pair, Salt Probing requires fewer keys by operating in the public-key setting. While known public-key cryptographic primitives are too computationally expensive to execute at line rate, we bypass this problem by using techniques reminiscent of TESLA[7] [23], so that public-key operations are used very infrequently. Hence, the computational overhead of Salt Probing is almost identical to that of Pepper Probing. Like TESLA, Salt Probing requires that Alice and Bob coarsely synchronize their clocks. Time is divided into *salt intervals*. In each interval Bob uses the *same salt value* to key each probing session with each Alice network.

Setup: Alice has some local secret key k_A, which is not shared with anyone; she uses this key to verify data that she sends to Bob and then is sent back to her. Bob has a public key PK_B that is known to Alice, and he keeps secret the corresponding secret key SK_B. Alice and Bob also know a fixed time constant a, which approximately equals 1.5 round trip times (RTT) (*e.g.*, $a = 150$ ms). Before the protocol begins, *Alice synchronizes her clock so that it lags Bob's clock by at most a seconds* as follows: At time t_A (on Alice's clock) Alice sends Bob a message $[A, t_A]_{k_A}$ MAC'd using her local secret key. Bob receives this message at time t_B (on Bob's clock) and responds with digitally signed message $[B, t_B, [A, t_A]_{k_A}]_{SK_B}$. (Note that a **digital signature** performs the same function as a MAC, but in the public-key setting; here the key SK used to sign a message is secret while the key PK used for verification is publicly known.) Alice will accept Bob's time t_B as long as his message is validly signed, contains a valid copy of Alice's message $[A, t_A]_{k_A}$, and arrives within a seconds of time t_A (on Alice's clock). Then, at time $t_A + a$ Alice synchronizes her clock to Bob's time t_B.

[7]TESLA uses only symmetric-key operations (except if a PKI is used for key exchange). Salt Probing could also be adapted to TESLA's symmetric key setting by adopting TESLA's use of one-way chains [23].

packet digest	ack	Probe
$z_1 = h(d_1)$		Yes
$z_2 = h(d_2)$	$[B, z_2, u,]_{\mathsf{s}_2(u)}$	Yes
$z_3 = h(d_3)$		No
$z_4 = h(d_4)$		No
$z_5 = h(d_5)$		No
$z_6 = h(d_6)$		No
$z_7 = h(d_7)$		No
$z_8 = h(d_8)$	$[B, z_8, u]_{\mathsf{s}_2(u)}$	Yes
$z_9 = h(d_9)$		No
$z_{10} = h(d_{10})$		No
$z_{11} = h(d_{11})$	$[B, z_{11}, u]_{\mathsf{s}_2(u)}$	Yes
$z_{12} = h(d_{12})$		Yes
$z_{13} = h(d_{13})$		No

Figure 6: Alice's table after at the end of salt interval u. Here Alice observes faults during exchanges $1, 12$.

If, after many attempts, Alice fails to receive a valid response to her synchronization message, then she decides the data path is faulty and raises an alarm. Synchronization happens infrequently (say, once a day).

Bob's algorithm: At the beginning of each salt interval u, Bob randomly chooses a pair of *salt* values $(\mathsf{s}_1(u), \mathsf{s}_2(u))$ that he keeps secret for the duration of the salt interval. Then, Bob runs a slightly modified version of Pepper Probing, replacing the symmetric key $k = (k_1, k_2)$ in Figure 4 and Equation 1 with the salt values $(\mathsf{s}_1(u), \mathsf{s}_2(u))$. That is, during salt interval u, Bob runs the Probe function of Equation 1 on packet digests $z = h(d)$ using salt $\mathsf{s}_1(u)$ as a key, and for each packet that is a probe, he constructs and sends to Alice an ack of the form $[B, z, u]_{\mathsf{s}_2(u)}$, which contains a packet digest z and a salt interval number u. As shown in Figure 5, Bob reveals the salt $(\mathsf{s}_1(u), \mathsf{s}_2(u))$ to Alice (and to all other source networks connected to him) a seconds after salt interval u ends by sending a (public-key) signed *salt release message* $[B, u, \mathsf{s}_1(u), \mathsf{s}_2(u)]_{SK_B}$.

Alice's algorithm: Because Alice does not know the salt for the duration of the salt interval, she is unable to run Probe 'in real time' as she sends each packet (as she did in Pepper Probing). Instead, Alice will store a packet digest for *each* of the packets that she sends to Bob as shown in Figure 6. Whenever Alice receives an ack $[B, z, u]_{\mathsf{s}_2(u)}$ from Bob, she stores the ack in her table only if the ack is tagged with the current salt interval u and a packet digest z that matches a packet digest that she has stored in her table. If Alice fails to receive a validly signed salt release message $[B, u, \mathsf{s}_1(u), \mathsf{s}_2(u)]_{SK_B}$ after salt interval u ends, she concludes that her data path to Bob is faulty and raises an alarm. Otherwise, Alice uses salt $\mathsf{s}_1(u)$ (from the salt release message) to run the Probe function on the packet digests in her table, and salt $\mathsf{s}_2(u)$ to verify the acks in her table. Then, to compute an estimate F of the fault rate that occurred during the salt interval, Alice counts

	Per-Packet Ack	**Pepper**	**Salt**	**Stealth** [4]
communication overhead (due to acks)	100%	p	p	p
key structure	symmetric	symmetric	public	symmetric
clock synch between Alice and Bob?	No	No	Coarse	No
modifications to Alice's traffic?	No	No	No	Yes
minimum duration of probing session, in packets	1	$\frac{64\beta}{p(\beta-\alpha)^2}$	$\frac{1}{q}\frac{64\beta}{p(\beta-\alpha)^2}$	$\frac{64\beta}{p(\beta-\alpha)^2}$
number of packet digests stored at Alice	all	p-fraction	q-fraction	p-fraction

Table 2: Comparison of (α, β)-secure FD protocols.

the fraction of exchanges for which $\mathsf{Probe}(z) =$ Yes and no valid ack is stored in her table. Finally, Alice raises an alarm if $F > \frac{\alpha+\beta}{2}$ and decides that everything was normal if $F < \frac{\alpha+\beta}{2}$.

A note on timing: Note that because Alice's clock lags Bob's clock by at most a seconds, it follows that there will be period of time of length $< a$ where Alice is operating in salt interval $u-1$ while Bob has already moved into salt interval u. To remedy this, during the first a seconds of each salt interval, Bob uses *both* the salt of the current interval $\mathsf{s}(u)$ and the salt from the *previous* interval $\mathsf{s}(u-1)$ in order to create his acks.

Security: The security of Salt Probing relies on (i) Eve's inability to skew Alice's synchronization to Bob's clock beyond a seconds, (ii) Eve's inability to forge a salt release message, and (iii) Eve's inability to bias Alice's estimate of the fault rate during a salt interval. To do (i), Eve needs to forge Bob's digital signature on his synchronization reply $[B, t_B, [A, t_A]_{k_A}]_{SK_B}$; notice that Alice will reject the synchronization reply if Eve delays it for more than a seconds. Furthermore, Eve cannot replay an old synchronization reply because the reply explicitly includes Alice's synchronization request. In order to do (ii), Eve needs to forge the digital signature on the salt message $[B, u, \mathsf{s}_1(u), \mathsf{s}_2(u)]_{SK_B}$. To see why (iii) holds, notice from Figure 5 that $\mathsf{s}(u)$ is known *only to Bob* for the duration of salt interval u on Alice's clock. As such, to bias Alice's fault estimate during salt interval u, Eve needs to break the security of Pepper Probing. As in Pepper Probing, Salt Probing is (α, β)-secure if Alice runs Salt Probing on at T packets where T is greater than the bound in Equation 2.

Efficiency: We compare Salt Probing, Pepper Probing, Per-Packet Ack and Stealth Probing [4] in Table 2. In contrast to Pepper Probing, in Salt Probing, Bob does not need to perform a key lookup for each packet he receives, since the same salt is used for each of Bob's probing sessions with each different Alice source network. The communication overhead and complexity of cryptographic operations for Salt Probing is almost identical to that of Pepper Probing (with the addition of a few infrequent public-key signature operations). Furthermore, while on the surface it may seem that in Salt Probing Alice needs to store information about each packet she sends to Bob for the duration of a salt interval, storage requirements can be reduced without compromising security if Alice *independently subsamples* packets with (truly random) probability q, as long as the total number of exchanges T that Alice uses for her estimate is a factor of $\frac{1}{q}$ greater than the bound in Equation 2. (See [1] for proof.)

For reasonable parameters: $p = 0.02$, false-alarm threshold $\alpha = 0.005$, and detection threshold $\beta = 0.01$, it follows from Equation 2 that Alice must store at least $qT = 1.3 \times 10^6$ entries in her table (Figure 6). If each packet digest $z = h(d)$ is 128 bits long, Alice must store on average about $128(1 + p) + 1 = 131$ bits for each row of her table, she requires about $qT \times 131 \approx 170$ Mbits of storage. Now, assuming that average packet is 3000 bits long, RTTs are on the order of 100 ms, and line rates are 5 Gbps, then about 10^7 packets are sent during one RTT. Then, if a salt interval lasts for about 5 RTTs, it follows that it is sufficient for Alice to subsample packets at rate $q = \frac{1.3\times10^6}{5\times10^7} \approx 2.6\%$ in order to obtain an unbiased measurement during a salt interval.

4. FAULT LOCALIZATION

In fault localization (FL) Alice must not only detect if her path to Bob is faulty, but she must also localize the faults to a particular link (or set of links) where she suspects the faults occurred. In this section, we first define security for per-packet FL and present a set of negative results that show the resources required for *any* secure FL protocol. We present an *Optimistic Protocol* for secure per-packet FL that leverages the useful cryptographic technique of *onion reports.* After discussing security for statistical FL, we show why the most natural technique for obtaining statistical FL from statistical FD protocols (where the source runs a probing protocol with each of the intermediate nodes on the path simultaneously [5, 21]) is vulnerable to a timing attack. Finally, we present a statistical FL protocol composed by having each intermediate node simultaneously run Salt or Pepper Probing with the destination network.

4.1 Per-Packet FL: Security Definition

Notation: Recall that a node can be either an AS or an individual router. We denote Alice, Bob, and the N intermediate nodes by R_A, R_B, $R_1, \ldots, R_N$. We say that the direction towards Alice is "upstream" and the direction towards Bob is "downstream". We say an *exchange* is all the communication associated with

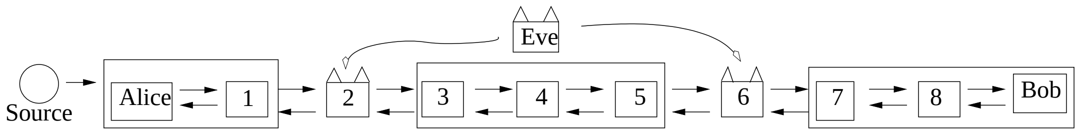

Figure 7: Fault localization security game.

sending a single data packet d.

Localizing links, not nodes: In the adversarial setting, it is impossible for Alice to always localize a *node* controlled by Eve. To see why, refer to Figure 7, where Eve controlling R_2 can always either: (a) become unresponsive by ignoring all the packets she receives from R_1, or (b) pretend that R_3 is unresponsive by dropping all communication to and from R_3. Notice, however, that case (a) could also have been caused by R_1 refusing to send packets to R_2, and case (b) could also have been caused by R_3 becoming unresponsive. It follows that at best, a fault localization protocol can pinpoint a *link* that is adjacent to a malicious node(s), rather than the malicious node itself. As such, in Figure 7, it suffices for Alice to localize the fault in case (a) to link $(1, 2)$ and the fault in case (b) to link $(2, 3)$.

Assumptions: We model and design FL protocols for the setting of *source routing* with *symmetric paths.* However, these assumptions only make our negative results stronger; a secure FL protocol that operates in a more general setting (where Alice does not know the identity of the nodes on the path) requires at least as much overhead as a secure FL protocol designed for source routing with symmetric paths.

Definition 4.1 (Per-packet FL Security)**.** The FL game is shown in Figure 7, for a path of length $N = 8$. Alice and Bob are connected to each other by a series of N intermediate nodes, some of which may be controlled by Eve. Eve is allowed to "corrupt" any set of nodes along the path (this models multiple adversarial nodes that collude), as in Figure 7 where she corrupted R_2 and R_6. She interacts with the honest nodes via the nodes that she corrupts. In Figure 7 we box the honest nodes together to emphasize that Eve cannot see or directly control any interactions between honest nodes. We assume that during the game, each link can drop packets due to *congestion* with some probability. The game consists of the Source sending data, and Eve playing with the packets that she sees (at each of the nodes she controls) until she decides to stop the game. At the end of the game, Alice uses the history of messages she received to output *a list of packets* that experienced faults, *and the links* where she believes these faults occurred.

Correctness condition: We require that if all the nodes are honest and there is no congestion, then Bob should be able to recover the data Alice sent, and Alice should never detect any faults. If congestion occurs, then Alice should localize the congested link.

Security condition: *Eve wins the FL game* if there is an exchange for which (1) Eve causes a fault that Alice does not detect, or (2) Eve causes Alice to implicate an un-congested link that is not controlled by Eve.

An FL protocol is secure if no efficient Eve is able to win the FL game with non-negligible probability.

Congestion: Since FL localizes faults to particular links, we want to accurately attribute faults, even ones caused by normal congestion, to the links on which they occurred. In order to do this, we need to explicitly model the congestion on every link, even those not adjacent to Eve. As such, we incorporate the probabilistic model of congestion in the game setting.

Monotonicity: In this case, our security definition does *not* imply monotonicity (as defined in Section 3.1), but in [1] we discuss how our protocols can be made monotone by including additional MACs or signatures.

4.2 Negative results

The security guarantees provided in FL are strictly stronger than those provided in FD; it follows that FL protocols require strictly more overhead than FD protocols. Here we show that, even in the setting of source routing with symmetric paths, in any secure per-packet FL scheme (1) Alice requires shared keys with Bob and the intermediate nodes, (2) Alice, Bob and each intermediate node must perform cryptographic operations, and (3) Alice and each intermediate node must modify storage for each packet sent. All these results hold in the statistical FL model of Section 4.4 (except that the result on the necessity of cryptography has some technical caveats for the statistical setting, see [1]), and outside the setting of source routing with symmetric paths. Again, our results on the necessity of keys and cryptography rely on the assumption that Eve actively forges acks, while our result on the necessity of storage holds even when Eve can only selectively drop packets.

Keys are necessary *at every node*: We claim that in any secure FL scheme, each node R_i must share secret information with Alice. We prove this in the contrapositive. Suppose there exists a node R_i who shares no secrets with Alice. Then consider an adversary Eve who controls all the nodes $1, \ldots, i-1$, illustrated in Figure 8, where $i = 2$ and we represent R_2's sad lack of keys with a frowney. In case (a) of Figure 8, R_3 is unreachable so Alice localizes the fault to link $(2, 3)$. However, because Alice shares no secrets with R_2, Eve

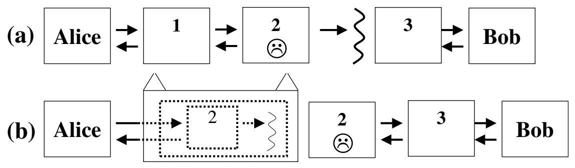

Figure 8: Necessity of keys and crypto.

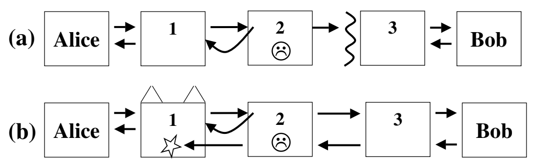

Figure 9: Necessity of storage.

knows everything that Alice knows about R_2! It follows that in case (b), Eve can drop all packets, and simulate R_2's behavior in a way that convinces Alice that R_3 was unreachable. Case (a) and (b) are indistinguishable for Alice. As such, in case (b) Alice will localize the fault to link $(2,3)$, which is not adjacent to Eve, and the FL protocol cannot be secure.

Cryptography is necessary *at every node*: We can also prove that the keys at each node must be used "in a cryptographically-strong manner". The structure of this proof resembles the structure of our previous proof; refer again to Figure 8, where this time $i = 2$'s frowney represents the sad fact that R_i does not perform cryptography. Again we will show that Eve controlling R_1 can break security by simulating R_2 in a manner that convinces Alice that link $(2,3)$ failed. However, now R_2's key is kept secret from Eve, so Eve needs to find some way to *learn* R_2's key in order to simulate R_2. We note that case (a), where R_3 is unreachable for the duration of an exchange, will occur relatively frequently during the course of the security game due to *congestion*. Because node R_2 does not use cryptography, each time case (a) occurs Eve at node R_1 *observes* the way R_2 reacts to congestion, and uses her observations *learn* 'a part of node R_2's secret key'. After seeing sufficiently many examples of case (a), Eve has learned enough to be able to simulate R_2's reaction to congestion and break security as in case (b). Formalizing this intuition is quite subtle and the details appears in [1]. The algorithm used to learn the key is a variant of the learning algorithm of [19], and the proof is an oracle separation in the blackbox model of [15].[8]

Modifying storage *per monitored packet* is necessary *at every node*: We prove in the contrapositive, using a timing attack, that each node must modify storage at least once for each packet it monitors. Suppose that we had an FL scheme in which node i does not modify storage when transmitting some packets. It follows that each time node R_i receives a message about packet d, he must immediately compute and send his response(s). Refer now to Figure 9 and where $i = 2$. Node R_2's sad lack of storage means that upon receipt of a message corresponding to packet d from R_1, he must immediately transmit a response to downstream node R_3 and he may also need to immediately send a response back upstream to node R_1 (we represent this by the arc in Figure 9). In case (a) R_2 is unreachable. Notice that since R_2 does not access storage, R_2 transmits no further messages corresponding to packet d because *he forgot that he is waiting for a response from* R_3! In case (a) Alice localizes the fault to link $(2,3)$. Now consider case (b), where Eve at R_1 forwards R_2's immediate response to her messages (again represented by the arc) but drops any communication that is not an immediate response to her message. From Alice's point of view, case (b) will look exactly like case (a), so Alice will implicate innocent link $(2,3)$, and the FL protocol cannot be secure.

[8]This negative result does *not* rule out non-blackbox constructions where intermediate nodes do not perform cryptography; however all known practical cryptographic constructions are blackbox, so for all practical purposes, this does not weaken our results. A full discussion of the implications of the blackbox nature of this result appears in [1].

4.3 Per-packet FL: Optimistic Protocol

Here we sketch our *optimistic per-packet FL protocol*, which can be thought of as a secure version of the Packet Obituaries [2] protocol. (Our protocol corrects a security vulnerability in [2]). Our presentation here is short and informal, but [1] has a full treatment, and also a technique to run this protocol in the asymmetric-path setting. This protocol is instructive as an example of a secure per-packet FL protocol with little additional communication overhead beyond Per-Packet-Ack, and because it demonstrates the use of *onion reports* (that appear again in our statistical FL protocol, Section 4.5).

We assume that each node R_i shares a MAC key k_i with Alice. For each packet that Alice sends, the protocol proceeds in two phases:

The send phase: This is identical to Per-Packet-Ack: Alice sends data packet d to Bob and Bob responds with ack $a = [B, h(d)]_{k_B}$. As before, Alice needs to store a data packet digest $z = h(d)$ and a time-out, but now we also require that all intermediate nodes store a digest of the data packet and ack that they see.

The report phase: This phase is run only if Alice detects a missing or invalid ack. Alice sends an onion report request to Bob, which tells all nodes along the path to prepare an onion report. To respond to the request, each router i creates an onion report θ_i. Each report θ_i contains the digest of the data packet and ack corresponding to the faulty exchange, which is concatenated to its downstream neighbor's onion report θ_{i+1} to form $\theta_i = [i, \text{data-digest}, \text{ack-digest}, \theta_{i+1}]_{k_i}$. We call these onion reports because each report layer is wrapped with a MAC and then included in the next layer. In [1] we prove that, because uncorrupted nodes do not modify the data or acks, all of the uncorrupted nodes immediately downstream of Alice must send "good reports" (defined formally in [1]). Thus, when Alice receives the completed report θ_1, checks to see where the first "bad report" appears, and implicates the upstream-most point where reports transition from "good" to "bad" as the faulty link.

Security: First, observe that since onion reports are MAC'd in layers that cannot be split apart, Eve cannot implicate a link between two innocent downstream nodes by selectively dropping reports. (Selective dropping of reports and acks is a significant vulnerability of the FL protocols of [2,5,21].) Now, observe that if Eve wins by security condition (1) (see Section 4.1) then she either must convince Alice that no fault occurred at all (in which case she has to break the security of the Per-Packet-Ack protocol of Section 3.3), or (2) she has to forge part of the onion report θ_1 in order to make it looks like the fault that she caused occurred at a link between two honest nodes, and not at a link adjacent to Eve. However, to modify the onion report so that it looks like a fault occurred between two honest nodes, Eve must be able to forge the MAC of an honest node. From the security of MACs and the Per-Packet-Ack protocol, it follows that Eve cannot win at by either security condition (1) or (2) with better than negligible probability, so that the FL protocol is secure.

4.4 Statistical FL: Security Definition

We now explore secure statistical FL. We define security for statistical FL, and discuss methods to construct statistical FL protocols out of secure FD protocols.

Naive sampling is not useful: In contrast to the sampling technique we used to build secure statistical FD protocols, simply having Alice sample some packets as probes and running FL on only those probe packets does not make FL much more efficient. To see why, observe that Alice cannot inform intermediate nodes which packets she designates as probes (if she did, a node occupied by Eve could then violate probe indistinguishability, see Section 3.4). It follows that intermediate nodes must behave as through every packet they see is a probe, and the storage overhead at the intermediate nodes remains as in a per-packet FL protocol!

Alarm thresholds for each link: In the spirit of (α, β)-security (see Definition 3.2), in statistical FL Alice should avoid raising an alarm when the fault rate is below the false-alarm threshold α, but if Eve drops more than a β-fraction of packets, then Alice should be able to localize at least one link adjacent to Eve. We can take an overall false-alarm threshold α and calculate per-link thresholds α_ℓ (and likewise for β_ℓ).[9] Alice should not complain if link ℓ experiences fault rate below α_ℓ, and she should raise an alarm if the fault rate exceeds β_ℓ.

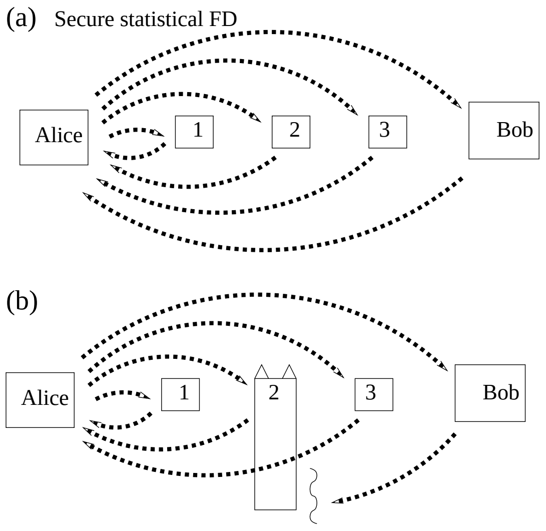

Figure 10: An insecure composition.

Definition 4.2 ((α, β)-statistical FL security)**.** We reuse the game setting of per-packet FL of Figure 7. The game consists of the Source sending *at least T packets* and then Eve playing with the packets that she sees at each of the nodes she controls. Let κ_ℓ denote the *true* fault rate along link ℓ, and let $\kappa = \sum_\ell \kappa_\ell$. At the end of the game, Alice outputs a list of links ℓ where Alice believes that $\kappa_\ell > \beta_\ell$.

Correctness and security conditions: Here we combine the correctness and security conditions, since it may be in Eve's interest to increase Alice's estimate of the fault rate on links between two honest nodes. We require that (1) for every link ℓ not adjacent to Eve and where $\kappa_\ell \leq \alpha_\ell$, Alice never outputs ℓ, and (2) if $\kappa > \beta$ (which only happens if Eve is acting maliciously) then Alice outputs at least one link ℓ that is adjacent to Eve.

A statistical FL scheme is secure if no efficient Eve can break the correctness and security conditions with non-negligible probability.

An insecure composition: Perhaps the most natural way to compose statistical FD protocols to obtain a statistical FL protocol (similar to the approach of [5,21], see Figure 10-(a)) is to have Alice run N simultaneous probing protocols with each of the intermediate nodes,

[9]There are several possibilities for calculating α_ℓ, β_ℓ. One is to assume that α is the overall honest congestion rate, and then compute the corresponding per-link honest congestion rates. For link $\ell = (i, i+1)$, this false-alarm threshold is $\alpha_\ell = (1-\alpha)^{i/N}(1-(1-\alpha)^{1/N})$ and the corresponding detection threshold is $\beta_\ell = \alpha_\ell \cdot (\beta/\alpha)$. In [1], we also present a more precise security definition where Alice gets an accurate *estimate* the fault rate for each link.

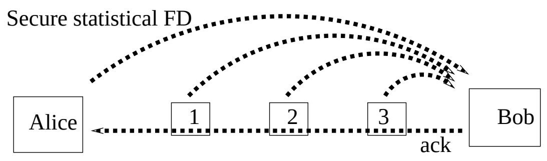

Figure 11: A secure composition.

and use the statistics from each to infer behaviour at each link. Unfortunately, this composition is vulnerable to the following *timing attack*: Suppose an isolated burst of packets travels through the network. The burst will trigger a separate burst of acks as it reaches each intermediate node. It follows that Eve (as in Figure 10-(b)) can determine the origin of each ack by observing the *time* at which the ack arrives. Then, Eve can drop all the acks originating from an honest node (Bob), causing Alice to implicate a link $(3, B)$ which is not adjacent to Eve. Notice that this attack results from the structure of the composition in Figure 10, and cannot be prevented even when acks are encrypted.

4.5 A Secure Composition for Statistical FL

We can use secure statistical FD protocols to construct a secure statistical FL via the composition method in Figure 11. We assume that every node R_i has public key PK_i that is known to everyone, and keeps secret the corresponding secret key SK_i. Each intermediate node runs a probing session with Bob, with one modification: whenever Bob decides to send an ack, Bob will address that ack to Alice *no matter whom the ack is destined for!* Each node forwards all acks upstream, and processes only the ack he expects. At the end of the FL session Alice will send a request $[A]_{SK_A}$, similar to the onion request of Section 4.3, to all the intermediate nodes. Then, upon node i's receipt of the onion report θ_{i+1} from downstream node $i+1$, node i responds with a digitally-signed onion report $\theta_i = [i, V_i, \theta_{i+1}]_{SK_i}$ where V_i is his *count* of the number of faults that occurred between himself and Bob.[10] When Alice receives the onion θ_1, she computes $F_\ell = \frac{V_i - V_{i+1}}{pT}$ for each link $\ell = (i, i+1)$, and adds ℓ to her list of faulty links if $F_\ell > \frac{\alpha_\ell + \beta_\ell}{2}$.

Security: Notice that now Eve cannot implicate an innocent link via (i) targeted dropping of acks, as in Figure 10-(b), because acks destined for different nodes are indistinguishable, or (ii) targeted dropping of reports, because the reports are onionized as in Section 4.3. Because the full proof is rather complex, we omit even a sketch here and refer the reader to [1].

[10]To prevent replay attacks, each node should also timestamp his onion report. To make the protocol monotone, each intermediate node R_i should only include the onion report from his downstream neighbour θ_{i+1} in his own report θ_i if the digital signature on θ_{i+1} is valid.

Choice of probing protocol: The composition works with either Pepper or Salt Probing. With Salt Probing, the communication overhead and the computations at Bob required for the composition are *identical* to those required for a single Salt Probing session between Alice and Bob, because the same set of packets are designated as probes (by Bob) for each intermediate node. Meanwhile, Pepper Probing increases communication overhead and processing at Bob, but less storage is required at Alice and the intermediate nodes.

Implications for routing architectures: We observe that architectures in which source networks make routing decisions by probing to intermediate nodes are also vulnerable to the timing attack in Section 4.4. Our results imply that architectures in which intermediate nodes probe to destination networks are preferable from the perspective of security.

5. CONCLUSIONS AND IMPLICATIONS

We conclude with some thoughts about the implications of our results on network architecture.

Fault Detection? Because FD protocols require only pairwise participation from nodes, deployment of FD can proceed in an incremental fashion that is compatible with incentives for informing routing decisions at the network edge. However, when we consider the placement and selection of FD protocols, natural questions arise about the division of labour between the end host and the edge router. We argue that the placement of FD protocols depends on the parties responsible for providing confidentiality and driving routing decisions. Consider, for example, the following three scenarios: (1) Firstly, if the end host both provides confidentiality and makes routing decisions, then host-based cryptography, *e.g.,* SSL, is appropriate for detecting faults on a per-packet basis. This may be the case in a new routing architecture, or a special-purpose secure network. (2) On the other hand, if the edge router both provides confidentiality and makes routing decisions, then Stealth Probing [4]'s secure statistical FD protocol, that also performs edge-to-edge encryption of traffic, is most appropriate. This particularly appropriate when two stub ASes belong to the same institution. (3) Finally, when routers make routing decisions but confidentiality is optionally provided at the end hosts, our Pepper and Salt Probing protocols (designed to avoid modifying and re-encrypting traffic at the edge router) are most appropriate. We believe that this is the case for the majority of scenarios at the edge of the Internet (*e.g.,* a residential broadband provider who runs FD on traffic that users optionally protect with SSL), as well as in the *core* of Internet. In fact, our Pepper and Salt Probing may even be efficient enough to be deployed in the core of Internet, as part of an architecture where core routers inform their routing decisions by running FD to destination networks.

Fault localization? FL protocols are likely to be practical for special settings, *e.g.,* highly-secure networks owned by a single party, or for a subset of packets within an AS, *e.g.,* network-management traffic. However, because FL requires active participation from the intermediate nodes, basing a network accountability framework on FL is not obviously compatible with the deployment incentives of the intermediate nodes on the Internet. (Recall that FL requires participation, *i.e.,* dedicated storage, at each intermediate node even in a weaker security model where Eve can only selectively drop packets.) Our negative results imply that reducing the complexity of FL is only possible under a weaker security model: for example, by assuming some trust between ASes, or by detecting bad behaviour by using additional out-of-band information about the content of the packets, or by considering an adversary with strictly weaker abilities. We leave a rigorous treatment of these alternatives for future exploration.

Accountability? Our FD protocols may be a more appropriate building block for constructing an accountability framework based on bilateral business relationships between ASes. For instance, a stub AS may blame poor edge-to-edge performance on its immediate provider who could, in turn, ascribe blame to the next AS in the path, as suggested in [16]. (Interestingly, the structure of our statistical FL protocol echos that of [16]'s accountability framework). Another possible accountability framework could combine multiple FD measurements from different vantage points to localize faulty links and nodes in the network, in the style of network tomography. However, traditional approaches to network tomography based on maximum likelihood estimation [6, 7], do not apply when an adversary is actively trying to evade detection. We believe that this open problem would benefit from a rigorous approach, similar to the one we took in this paper, that combines theoretical cryptography with statistics to construct secure protocols and establish negative results for the adversarial setting.

6. REFERENCES

[1] Measuring path quality in the presence of adversaries: The role of cryptography in network accountability. *Technical Report*, Available upon request.

[2] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing packet obituaries. *ACM HotNets-III*, 2004.

[3] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. In *IEEE INFOCOM*, 2004.

[4] I. Avramopoulos and J. Rexford. Stealth probing: Efficient data-plane security for ip routing. *USENIX*, 2006.

[5] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *ACM WiSE*, 2002.

[6] R. Caceres, N. Duffield, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal loss characteristics. *IEEE Trans. Info. Theory*, 45(7), 1999.

[7] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: Recent developments. *Statistical Science*, 19(3):499–517, 2004.

[8] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's Internet. *IEEE/ACM Trans. Netw.*, 13(3), 2005.

[9] M. Crovella and B. Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications.* Wiley, 2006.

[10] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *ACM SIGCOMM*, 2000.

[11] O. Goldreich. *Foundations of Cryptography.* Cambridge University Press, 2007.

[12] A. Herzberg and S. Kutten. Early detection of message forwarding faults. *SIAM J. Comput.*, 30(4):1169–96, 2001.

[13] IETF. Working Group on IP Performance Metrics (ippm). *http://www.ietf.org/html.charters/ippm-charter.html.*

[14] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. *FOCS*, 1989.

[15] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. *STOC*, 1989.

[16] P. Laskowski and J. Chuang. Network monitors and contracting systems: competition and innovation. In *ACM SIGCOMM*, 2006.

[17] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. User-level Internet path diagnosis. *SOSP*, 2003.

[18] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: detecting and isolating malicious routers. In *IEEE Internat. Conf. Dependable Systems and Networks*, 2005.

[19] M. Naor and G. N. Rothblum. Learning to impersonate. In *International Conf. on Machine Learning*, 2006.

[20] NIST. Hash function workshop. *http://csrc.nist.gov/pki/HashWorkshop/.*

[21] V. Padmanabhan and D. Simon. Secure traceroute to detect faulty or malicious routing. *HotNets-I*, 2002.

[22] R. Perlman. *Network Layer Protocols with Byzantine Robustness.* PhD thesis, MIT, 1988.

[23] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Security and Privacy Symposium*, 2000.

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

[25] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. H. Katz. Listen and Whisper: Security mechanisms for BGP. In *USENIX NSDI*, 2004.