Path-Quality Monitoring in the Presence of Adversaries

Sharon Goldberg[1], David Xiao[1], Eran Tromer[2], Boaz Barak[1], Jennifer Rexford[1]

[1] Princeton University, Princeton, NJ 08544
[2] MIT, Cambridge, MA 02139

ABSTRACT

Edge networks connected to the Internet need effective monitoring techniques to drive routing decisions and detect violations of Service Level Agreements (SLAs). However, existing measurement tools, like ping, traceroute, and trajectory sampling, are vulnerable to attacks that can make a path look better than it really is. In this paper, we design and analyze path-quality monitoring protocols that reliably raise an alarm when the packet-loss rate and delay exceed a threshold, even when an adversary tries to bias monitoring results by selectively delaying, dropping, modifying, injecting, or preferentially treating packets.

Despite the strong threat model we consider in this paper, our protocols are efficient enough to run at line rate on high-speed routers. We present a *secure sketching* protocol for identifying when packet loss and delay degrade beyond a threshold. This protocol is extremely lightweight, requiring only 250–600 bytes of storage and periodic transmission of a comparably sized IP packet to monitor billions of packets. We also present *secure sampling* protocols that provide faster feedback and accurate round-trip delay estimates, at the expense of somewhat higher storage and communication costs. We prove that all our protocols satisfy a precise definition of secure path-quality monitoring and derive analytic expressions for the trade-off between statistical accuracy and system overhead. We also compare how our protocols perform in the client-server setting, when paths are asymmetric, and when packet marking is not permitted.

Categories and Subject Descriptors. C.2.2 Computer Communication Networks: Network Protocols
General Terms. Measurement, Security.

1. INTRODUCTION

Path-quality monitoring is a crucial component of flexible routing techniques (e.g., intelligent route control, source routing, and overlay routing) that give edge networks greater control over path selection. Monitoring is also necessary to verify that service providers deliver the performance specified in Service-Level Agreements (SLAs). In both applications, edge networks need to determine when path quality degrades beyond some threshold, in order to switch from one path to another or report an SLA violation. The problem is complicated by the presence of nodes along the path who try to interfere with the measurement process, out of greed, malice, or just misconfiguration. In this paper, we design and analyze light-weight path-quality monitoring (PQM) protocols that detect when packet loss or delay exceeds a threshold, even when adversaries try to bias monitoring results. Our solutions are efficient enough to run at line rate on the high-speed routers connecting edge networks to the Internet.

SIGMETRICS'08, June 2–6, 2008, Annapolis, Maryland, USA.

1.1 The presence of adversaries

Today, path-quality monitoring relies on active measurement techniques, like ping and traceroute, that inject special "probe" packets into the network. In addition to imparting extra load on the network, active measurements are vulnerable to adversaries that try to bias the results by treating probe packets preferentially. Instead, we want to design protocols that provide accurate information even when intermediate nodes may *adversarially delay, drop, modify, inject or preferentially treat* packets in order to confound measurement. Our motivations for studying this adversarial threat model are threefold:

1. It covers active attacks. Our strong threat model covers a broad class of malicious behavior. Malicious adversaries can easily launch routing-protocol attacks that draw packets to (or through) a node of their choosing [6], or compromise one of the routers along an existing path through the Internet [15, pg. 14]. Biasing path-quality measurements allows the adversaries to evade detection, while continuing to degrade performance or impersonate the legitimate destination at will. In addition, ISPs have both the economic incentive and the technical means to preferentially handle probe packets, to hide discrimination against unwanted traffic like Skype [25] or BitTorrent [1], and evade detection of SLA violations. (In fact, commercial monitoring services, like Keynote, claim to employ "anti-gaming" techniques to prevent providers from biasing measurement results [2].) Finally, adversaries controlling arbitrary end hosts (such as botnets) can add "spoofed" packets to the stream of traffic from one edge network to another, to confound simplistic measurement techniques (e.g., such as maintaining a counter of received packets).

2. It covers all possible benign failures. By studying the adversarial setting, we avoid making *ad hoc* assumptions about

the nature of failures caused by normal congestion, malfunction or misconfiguration. Even benign modification of packets may take place in a seemingly adversarial manner. For example, an MTU (Maximum Transmission Unit) mismatch may cause a router to drop large packets while continuing to forward the small probe packets sent by ping or traceroute [21]. As another example, link-level CRC checks are surprisingly ineffective at detecting the kinds of errors that corrupt IP packets [29]. Since the adversarial model is the strongest possible model, any protocol that is robust in this setting is automatically robust to all other kind of failures.

3. It is challenging to satisfy in high-speed routers. We choose to work in a difficult space, where we assume the strongest possible adversarial model, and yet design solutions for high-speed routers on multi-Gbit/sec links, where computation and storage resources are extremely limited. We view it as an important research goal to understand what can and cannot be done in this setting, to inform practical decisions about what level of threats future networks should be designed to withstand. Furthermore, designing protocols for this adversarial setting is not simply a matter of adding standard cryptographic tools to existing non-adversarial measurement protocols. Indeed, naive ways of combining such protocols with cryptographic tools may be either insecure or very inefficient (*e.g.,* encrypting and authenticating all traffic).

Despite the strong threat model we consider in this paper, we are still able to design secure PQM protocols that can be implemented in the constrained environment of high-speed routers. Our protocols are competitive, in terms of efficiency, with solutions designed for the non-adversarial setting [12, 16] and for weaker threat models. As such, we believe that our protocols are strong candidates for deployment in future networks, even where our strong security guarantees may not be essential.

1.2 Our results

We say that a *packet delivery failure* (*failure* for short) has occurred on a path if a packet sent by the source was dropped, modified, or delayed beyond a certain timeout period, regardless of whether the drop is due to congestion, malfunction or adversarial behavior. The goal of a PQM protocol is to *detect* when the fraction of failures on a path rises above a certain fraction β (say $\beta = 0.01$) of all packets sent. We emphasize that a PQM protocol does not *prevent* failures. A *secure* PQM protocol achieves its goal even when there is an intermediate node on the path between source and destination that can adversarially drop, modify, or inject both data and protocol-related packets to the path in order to bias the measurement results. Most existing PQM protocols, such as ping, traceroute, and counter-based solutions [30] completely break down in this setting (we show why in Section 2.2).

To have efficient solutions that can run on high-speed routers, we design secure PQM protocols based on two main classes of data-reduction techniques:

Secure sketch. In Section 5, we present a protocol for monitoring packet-loss rates that makes extremely efficient use of communication and storage resources. Our secure sketch protocol uses ℓ_2*-norm estimation sketches* [3, 4, 9, 31] to aggregate information about the failures that occur during an interval, in which T packets are sent, into a *sketch* of size $O(\log T)$ bits; the communication overhead is just a single report packet per time interval. Assuming that about 10^7 packets are sent during an 100ms interval, our protocol requires between 250–600 bytes of storage at the source and destination, and a report can easily fit into a single IP packet. In the course of analyzing this protocol, we provide an improved formal analysis of the performance of [9]'s sketching scheme that may be of independent interest.

Secure sampling. In certain settings, an edge-network may require accurate round-trip delay measurements in addition to monitoring if the failure rate rises above a threshold. Section 4 describes a secure PQM protocol that achieves this by measuring performance for a sample of the traffic that is obtained using a cryptographic hash function. For PQM with threshold β, this sampling-based protocol requires $O(n/\beta)$ bits of storage at the source, where n is the output length of the hash function. We present two variants: (1) *Symmetric Secure Sampling* is designed for the setting where source and destination can devote an equal amount of resources to the running of the protocol, and (2) *Asymmetric Secure Sampling*, which is designed for a client-server setting where the client contributes the bulk of the resources, and the server participates in path-quality monitoring with many clients simultaneously.

Precise definition of security. Evaluating the security of a protocol is challenging in practice. In many problem domains, *e.g.,* intrusion detection, the only viable approach is to enumerate a set of possible attacks, and then show how the protocol defends against these specific attacks. One way to do this is to evaluate the protocol on, say, packet traces of real-world attacks. However, there is always a risk that an adversary might devise a new attack that we have not considered or that was not expressed in the trace. Fortunately, in our problem domain, a more comprehensive security evaluation is possible. Namely, instead of enumerating ways the protocol can break down (*i.e.,* attacks), we can instead give a precise definition of the functionality we require from the protocol, and then guarantee that the protocol can carry out these functions *even in the face of all possible attacks by an adversary* with a specific set of powers.

To do this, in Section 2 we precisely define our requirements for a secure PQM protocol and the powers that we give to the adversary. Then, to evaluate the security of our protocols, we use formal analysis to *prove* that our protocols achieve this functionality *no matter what* the adversary does, short of breaking the security of the basic cryptographic primitives (*e.g.,* digital signatures and hash functions) from which the protocol is constructed. In Section 6 we prove that *any* secure PQM protocol (as per Definition 1) would need to employ the same basic security machinery—secret keys and cryptographic operations—used by our secure sketching and sampling protocols.

Evaluating performance. The performance and cost of any particular implementation of our protocols would depend on memory speed and the particular choice of cryptographic primitives. As such, we count separately the different resources—computation, storage and communication—used by our protocols, bound the resource utilization using formal analysis, and also show somewhat better bounds through numerical experiments. Our protocols use cryptographic hash functions in an *online* setting, where an adversary has very limited time to break the security before

the hash parameters are refreshed; this allows us to use fast implementations of these hash functions (details in the full version [14]). We emphasize that all except one of our protocols *do not modify data packets* in any way, and so they may be implemented off the critical packet-processing path in the router. Not marking packets also makes our protocols backwards compatible with IP while minimizing latency at the router, allows the parties to turn on/off PQM protocols without the need to coordinate with each other, and avoids problems with increasing packet size and possibly exceeding the MTU. For efficiency reasons, we specifically avoid solutions that require encryption and authentication of all the traffic sent on the path, as in IPsec. We further discuss and compare the performance trade-offs for our sketch and sampling protocols with known solutions like IPsec in Section 7.

2. THE STATISTICAL SECURITY MODEL

In our model, a source *Alice* sends packets to a destination *Bob* over a path through the Internet. Fixing a particular time period, which we call an *interval*, we define a *packet delivery failure* to be any instance where a packet that was sent by Alice during the interval fails to arrive unmodified at Bob before the the interval ends. An adversary *Eve* can sit anywhere on the path between Alice and Bob, and we empower Eve to drop, modify, or delay every packet or add her own packets. A *path quality monitoring (PQM) protocol* is a protocol that Alice and Bob run to detect whether the number of failures during the interval exceeds a certain fraction of total packets transmitted.

DEFINITION 1. Given parameters $0 < \alpha < \beta < 1$ and $0 < \delta < 1$, we say a protocol is a (α, β, δ) *secure PQM protocol* if, letting T be the number of packets sent during the interval:

1. *(Few false negatives.)* If more than βT packet delivery failures occur then the protocol raises an alarm with probability at least $1 - \delta$, *no matter what Eve does.*
2. *(Few false positives.)* If no intermediate node behaves adversarially and at most αT failures occur then the protocol raises an alarm with probability at most δ.

We assume that the T packets sent during an interval are distinct, because of natural variation in packet contents, and the fact that even successive packets sent by the same host have different ID fields in the IP header [12] (note that even retransmissions of the same TCP segment correspond to distinct IP packets, because of the IP ID field).

2.1 Properties of our security definition

Our definition is strongly motivated by our intended application of enabling routing decisions or SLA violation detection. The most important security guarantee it provides is that *no matter what Eve does* she cannot prevent Alice from raising an alarm when the failure rate for packets that Alice sent to Bob exceeds β. As such, our definition encompasses attacks by nodes on the data path that include (but of course are not limited to): colluding nodes that work together in order to hide packet loss, an adversarial node that intelligently injects packets based on timing observations or deep packet inspection, a node that preferentially treats packets that it knows are part of the PQM protocol, and a node that masks packet loss by injecting an equal number of nonsense packets onto the data path. We emphasize that we never make any assumptions on the distribution of packet loss on the path; our model allows for any possible 'failure model', including one where, say, packet loss is correlated across different packets.

On the other hand, as a routing-decision enabling tool, we do not require PQM protocols to *prevent* packet delivery failures but rather only *detect* them. Second, rather than determining *exactly* how many failures occurred, the protocol is only required to detect if the number of failures exceeds a certain threshold. (While solutions that exactly count failures certainly exist, *e.g.,* see discussion on IPsec in Section 7, they typically require cryptographically authenticating and/or encrypting all traffic and hence are rather expensive to operate in high-speed routers.) Third, we do not require our protocols to distinguish between packet failures occurring due to adversarial tampering or due to "benign" congestion or malfunction.

Finally, while our security definition requires that our protocols do not raise a (false) alarm when the one-way failure rate is less than α for the *benign setting*, we do allow for the possibility of raising an alarm due to adversarial tampering even when fewer than an α fraction of failures occur. This is because an adversarial node has the power to arbitrarily tamper not just with data packets, but also with any packets that are sent as part of the PQM protocol; thus Eve can always make a path look worse by selectively dropping all acknowledgment or report messages that Bob sends to Alice, even if all the original packets that Alice sent to Bob were actually delivered. (In this paper, we will assume that any acknowledgment or report messages that Bob sends to Alice are sent repeatedly to ensure that, with high probability, they are not dropped due to normal congestion.) When this happens, it may very well make sense for the protocol to raise an alarm, and the router to look for a different path.

2.2 Related works

The literature on path-quality monitoring typically deals only with the benign setting; most approaches either have the destination return a count of the number packets he receives from the source, or are based on active probing (ping, traceroute, [17,27,28] and others). However, both approaches fail to satisfy our security definition. The counter approach is vulnerable to attack by an adversary who hides packet loss by adding new, nonsense packets to the data path. Active probing fails when an adversary preferentially treats probe packets while degrading performance for regular traffic, or when an adversary sends forged reports or acknowledgments to mask packet loss. Even known passive measurement techniques, where normal data packets are marked as probes, either explicitly as in IPPM [17] or implicitly as in Trajectory Sampling [12] and PSAMP [16], are vulnerable to the same attacks as active probing techniques if the adversary can distinguish the probe packets from the non-probe packets (*e.g.,* see [13] for attacks on PSAMP).

To obtain path-quality monitoring protocols that work in the adversarial setting, we have developed protocols that are more closely related to those used for traffic characterization. For example, our secure sampling protocol uses passive measurement techniques similar to those of [12,16], that are designed for characterizing traffic mix. Similarly, our secure sketch protocol draws on ℓ_2-norm estimation schemes [3, 4,

9, 31] that are typically uses to characterize data streams. (See *e.g.,* [32] for a survey of data streaming algorithms used in networking.) Because our protocols are designed for the adversarial setting, they require the use of keys and cryptographic hash functions (see sections 3 and 6) in order to prevent an adversary from selectively adding and dropping packets in a manner that skews the estimate returned from the sketch. On the other hand, we can use the special structure of the path-quality monitoring setting to prove new analytical bounds which result in *provably lower communication and storage requirements* than those typically needed in traffic characterization applications. Also, at the end of Section 5.3 we discuss how the new result of [23] for sketching adversarially-chosen sets could be applied to our setting.

Our results are also related to works in the cryptography and security literature. In the security literature, traditional works on providing availability typically give guarantees on a *per-packet* basis, resulting in schemes with very high overhead, see *e.g.,* [11] and later works. While *statistical* PQM protocols have been considered in the security literature [5, 24, 30], ours is the first work in this area to provide a formal security definition and to *prove* the security of our protocols within this model. We argue that such a model is crucial to understanding the security guarantees provided by a protocol. Indeed, one of Fatih's [24] PQM approaches is based on a simple counter (and is therefore vulnerable to the attack described above), while Listen [30] is a protocol that does not use cryptographic operations, and is thus vulnerable to attack by an intermediate node that injects false acknowledgments onto the path. Finally, while Stealth Probing [5] is secure in our model, it incurs the extra overhead of encrypting and authenticating all traffic.

3. CRYPTOGRAPHIC PRIMITIVES

Our PQM protocols use several cryptographic primitives, with different security properties and performance. We describe the security properties of these primitives below:

A *Collision-Resistant Hash (CRH)* function is a function H for which it is infeasible to find a collision, *i.e.,* $m \neq m'$ fulfilling $H(m) = H(m')$. Typical choices of H are SHA-1 and (truncated) SHA-256. The output of $H(x)$ is called the *digest* of x, and we assume it is 160 bits long.

A *PseudoRandom Function (PRF)* is a keyed function $h_k(\cdot)$ that maps an arbitrary length string to an n-bit string using a key k; in our case, $n = 64$ or 96 suffice. If the key k is chosen at random, then to an adversary with no knowledge of k the function $h_k(\cdot)$ looks totally unpredictable and cannot be distinguished (except with an exponentially small probability) from a truly random function (where each input is mapped independently to a uniformly random output). Hence, in our analysis we may treat h_k as if it is truly random. All our protocols require a PRF computation on the entire contents of every sent packet,[1] and all subsequent processing of the packet relies only on this hash value. In the full version [14] we argue that our *online setting* allows us to realize the PRF via fast cryptographic hash functions in *both hardware and software* that support multi-Gbit/sec packet streams.

[1]For convenience, we abuse notation and say that whenever the PRF is applied to a packet, the non-invariant fields of the packet header are discarded from the input. In the case of IPv4, this means excluding the ToS, TTL and IP checksum (see [12, Section II.A]).

A PRF can be used to realize a *Message Authentication Code (MAC)*: using a shared key k, for a message m, one party will send $(m, h_k(m))$ and the other party can verify that a pair (m, t) satisfies $t = h_k(m)$. The value $h_k(m)$, called the *tag*, cannot be feasibly forged by an adversary that does not know k. We denote $\mathrm{MAC}_k(m) = (m, h_k(m))$. *Digital signatures* provide authenticity in the public-key setting. Here a private key SK is used to sign a message m and obtain a signature σ; we denote this with $\sigma = \mathrm{Sign}_{SK}(m)$. A public key PK is known to all parties and is used to verify the signature; the $\mathrm{Verify}_{PK}(\sigma)$ operation outputs a message m for valid signatures and aborts otherwise. Digital signatures are more computationally expensive than MACs, so we use them only for infrequent synchronization data.

Keys. While some of our protocols require parties to share a pairwise secret key, this does not imply that we must maintain an infrastructure of pairwise keys for the Internet. All of our protocols require participation of only two parties. Parties can derive pairwise keys via, *e.g.,* authenticated Diffie-Hellman key exchange (as used in TLS/SSL [10]) using Public Key Infrastructure such as DNSSEC or some out-of-band secure channel. Furthermore, an organization owning multiple routers running PQM might have an incentive to assign pairwise secret keys. Once a pairwise shared master key is established, keys for specific intervals and runs of the protocol can be derived locally at each party using a PRF h'. For example, we can use $k_u = h'_k(u)$ where k_u is the key for interval u, and k is the master key. Here, because the PRF h' is used only once per interval, and also needs to be resilient against many queries, we let h' be traditional conservative pseudorandom function such as AES-CBC-MAC.

4. SECURE SAMPLING

In a sampling-based protocol, Alice and Bob agree on a small set of packets (the probes) for which Alice expects acknowledgments from Bob. Then, Alice can detect when the path quality is unacceptable when too many probes are unacknowledged. These protocols limit the storage and communication overhead because only a small fraction of traffic is monitored, and also allow Alice to measure round-trip delay by monitoring arrival time of acks. However, such protocols are inherently vulnerable to adversaries that preferentially allow probes to travel unharmed, but drop, delay, or modify other packets. Since most packets are not probes, such an adversary can disrupt traffic without Alice realizing that something went wrong. To prevent such attacks, in our secure sampling protocols that Alice and Bob use a shared PRF to coordinate their sampling. The cryptographic properties of the PRF, discussed in Section 3, prevent an adversary from distinguishing probes from non-probes.[2] Use of a PRF in our setting is necessary for security; in the full version [14] we show an example of why a non-cryptographic hash function (*e.g.,* CRC) is insufficient.

We present three protocols. The Symmetric Secure Sampling protocol is designed for the setting where Alice and Bob share pairwise secret keys. The two Asymmetric Secure Sampling protocols (one for senders and one for re-

[2]We stress that probes are ordinary data packets that are part of the data stream and are not explicitly marked. Alteration of packets is undesirable for several reasons for example: it must be undone by the receiver prior to processing or forwarding, and it may run into MTU limitations, etc.

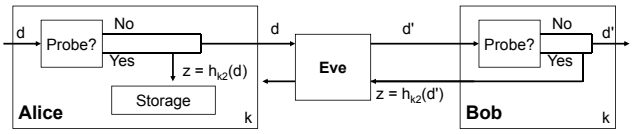

Figure 1: Secure Sampling.

ceivers) use a variant of *delayed-exposure techniques* (*c.f.,* TESLA [26] and the references therein) to eliminate the need for pairwise keys, at the cost of some increased storage at Alice or Bob. The asymmetric protocols are especially advantageous when one of the parties is a server that needs to engage in simultaneous PQM sessions with many clients.

4.1 Symmetric Secure Sampling

We assume Alice and Bob share a secret (master) key k. They also know a parameter p, called the *probe frequency*. During each interval, our symmetric secure sampling protocol operates as follows:

1. Alice and Bob derive an interval-specific secret key by applying a PRF keyed with the master key k to the interval number u, *i.e.,* $(k_1, k_2) = h'_k(u)$. In the full version [14], we give a detailed treatment of techniques that can be used to achieve interval synchronization between Alice and Bob.

2. After transmitting each packet d, Alice decides whether d is a probe. Specifically, she uses k_1 and the probe frequency p to run a Probe function that is implemented using a PRF h keyed with k_1 and outputting an integer in $\{0, \ldots, 2^n - 1\}$, as follows:

$$\mathsf{Probe}_{k_1}(d) = \begin{cases} \text{YES}, & \text{if } \frac{h_{k_1}(d)}{2^n} < p; \\ \text{NO}, & \text{else.} \end{cases} \quad (1)$$

   If $\mathsf{Probe}_{k_1}(d)$ outputs YES then Alice stores the tag $z = h_{k_2}(d)$ in a table.[3]

3. Bob receives d' and computes $\mathsf{Probe}_{k_1}(d')$. If it outputs NO then do nothing; if it outputs YES then transmit the tag $z' = h_{k_2}(d')$ back to Alice.

4. Alice receives the acknowledgment z' and removes it from her table (if it is present in her table).

At the end of an interval, Alice raises an alarm if and only if her table contains more than $pT\sqrt{\alpha\beta}$ remaining entries.[4] Otherwise she does not raise an alarm.

THEOREM 2. *The symmetric secure sampling protocol is an (α, β, δ)-secure PQM protocol for $\alpha < \beta \leq 4\alpha$ as per Definition 1, whenever the probe frequency p and number of packets per interval T satisfy*

$$pT > \ln(\tfrac{1}{\delta}) \tfrac{3}{(\sqrt{\beta}-\sqrt{\alpha})^2} \quad . \quad (2)$$

[3]When h uses a modified Wegman-Carter construction (see the full version [14]), the computation of $h_{k_2}(d')$ can reuse the universal hash already computed for $h_{k_1}(d')$, and thus amounts to a single AES or DES invocation.

[4]To obtain this threshold, we could have used the mid point between $p\alpha T$ and $p\beta T$. However to get much better parameters for our protocols, we can apply maximum likelihood estimation to obtain the threshold above, since (from proof of Theorem 2) V, or the number of unacknowledged probes in Alice's table, is a binomial random variable.

When $\alpha = \beta/2$ we can obtain a slightly better bound $pT > \ln(\frac{1}{\delta}) \frac{2\ln 2}{(\sqrt{\beta}-\sqrt{\alpha})^2}$, so that when $\delta = 1\%$, we require $pT > 75/\beta$.

PROOF OF THEOREM 2. First, we observe that *regardless of any strategy Eve adopts*, and independently of all other packets, the probability each dropped/modified packet is a probe is p. To see why, recall that we assumed that packets sent by Alice are unique. If $h_{k_1}(\cdot)$ in Probe were replaced by a truly random function, then every packet would be a probe independently with probability p. The same must hold for the real implementation of Probe using h_{k_1}, since otherwise Eve could distinguish between the PRF and a truly random function, contradicting the security of the PRF.

For the false positives condition of Definition 1, suppose the failure rate is less than α. The probability of misdetection is the probability that a larger than $\sqrt{\alpha\beta}$-fraction of the samples are dropped. Let V be the number of remaining (unacknowledged) entries in Alice's table. When each packet is independently sampled with probability p, then if $\beta < 4\alpha$ we can find the false positive probability using a Chernoff bound as

$$\Pr[\, V > pT\sqrt{\alpha\beta} \mid \text{failure rate } < \alpha \,] \leq e^{-\frac{(\sqrt{\beta}-\sqrt{\alpha})^2}{3} pT} \quad . \quad (3)$$

By our observation above, this inequality still holds (up to a negligible additive factor) when we sample probes using a pseudorandom function.

Next, consider the false negatives condition of Definition 1. First note that Eve cannot forge a valid ACK to a packet that was not received by Bob, since she only sees the output of the PRF h_{k_2} on packets that Bob receives, and cannot predict its value on any other input. Therefore all that Eve can do is to bias the measurement by preferentially dropping non-probes. again, if probes are sampled independently with probability p then

$$\Pr[\, V < pT\sqrt{\alpha\beta} \mid \text{failure rate } > \beta \,] \leq e^{-\frac{(\sqrt{\beta}-\sqrt{\alpha})^2}{2} pT} \quad . \quad (4)$$

As observed above, (4) still holds (up to a negligible factor) when the probes are sampled using a PRF. Notice that dropping ACKs cannot help Eve, as it only makes the source *more* likely to raise an alarm. It follows from equations (3), (4) and Definition 1 that, given α, β and δ, such that $\beta < 4\alpha$, the protocol is secure whenever (2) holds. □

4.2 Asymmetric Secure Sampling

This section describes variants of the above protocol for the case where a single router (the *server*) deals with a large number of other routers (the *clients*). Our protocols support server scalability by minimizing the per-client cost of the server. In particular, the server will not need to establish a separate key for every client. We will, however, assume that the clients can dedicate more resources to the PQM protocol. We provide two different protocols, depending on whether the server is receiving from, or sending to, its clients (of course, the two PQM protocols can be applied jointly to monitor both directions).

We again divide time into intervals, and the idea is that the server performs his operations (as either sender or receiver) with private keys, which we call the *salt*, unknown to anyone except himself until the end of the interval, at which time he releases the salt to all interested clients. The point is that by the time the server releases the salt it is too late to cheat; note that even dishonest clients cannot cheat

honest clients because no one except the server knows the salt until the end of the interval.

Instead of using symmetric keys between each pair of parties, here we assume that the server has a public/private key pair (PK, SK) where the public key PK is known to all parties (e.g., through a Public Key Infrastructure). To ensure that the computationally-expensive public-key operations are used infrequently, we will use cryptographic delayed-exposure techniques (*c.f.,* TESLA [26] and the references therein) that require secure clock synchronization. We assume that each client securely synchronizes her clock so that it lags behind the server's clock by at most τ seconds, where τ is a constant known to all parties. In the full version [14] we present a simple secure protocol for achieving this synchronization.

4.2.1 Receiving-Server Secure Sampling (RSSS)

We first consider the case where a single server (Bob) is receiving traffic from multiple clients (each playing the role of Alice). The following protocol allows every client to monitor the path quality for traffic that it sends to the server, while the server requires *no* storage and can use the same key to engage in PQM with every client. During the u-th interval, the RSSS protocol operates as follows:

1. *(Interval Setup.)* Bob, the receiver, randomly chooses a pair of salt values $(\mathsf{s}_1(u), \mathsf{s}_2(u))$ that he keeps secret until the very end of the interval.

2. *(Packet Transmission.)* Packet transmission during the interval proceeds as follows:

   - For each packet d Alice wishes to send, she stores the digest $H(d)$ in her table. Suppose Alice sends T packets in total. (This means Alice stores T digests. In Section 4.3 we discuss how Alice can independently subsample packets to reduce her storage requirements.)
   - Upon receiving each packet d', Bob computes its digest $z' = H(d')$. He then evaluates $\mathsf{Probe}_{\mathsf{s}_1(u)}(z')$; if NO then he does nothing, and if YES then he transmits an ACK of the form $\mathrm{MAC}_{\mathsf{s}_2(u)}(z', u)$ back to Alice.
   - Each sender (Alice) stores all the ACKs received which included the current interval u.

3. *(Salt Release.)* Bob maintains the secrecy of the salt until τ seconds after interval u ends. At that time he reveals the salt $(\mathsf{s}_1(u), \mathsf{s}_2(u))$ to all clients by sending a SALTRELEASE packet containing $\mathrm{Sign}_{SK}(u, \mathsf{s}_1(u), \mathsf{s}_2(u))$.

4. *(Security check.)* If Alice fails to receive a SALTRELEASE containing a signature σ within 1 RTT after the interval u ends, or if $\mathrm{Verify}_{PK}(\sigma)$ doesn't return a tuple $(u, \mathsf{s}_1(u), \mathsf{s}_2(u))$, then Alice raises an alarm. Otherwise, she uses salt $\mathsf{s}_1(u)$ 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 Alice counts the number of packets for which $\mathsf{Probe}_{\mathsf{s}_1(u)}(z) = \text{YES}$ and no valid ACK is stored in her table; call this count V. Finally, Alice raises an alarm if $V > pT\sqrt{\alpha\beta}$.

Notice that our protocol does *not* require Bob to send out the salt immediately at the end of the interval. However, we observe from Step 5 above, that there is a tradeoff between frequency of salt release messages and storage at Alice; the longer Bob delays sending out the salt, the longer Alice has to wait before she can clear her table.

Assume for now that all parties' clocks are perfectly synchronized. Then Eve cannot cheat within any single interval:

THEOREM 3. *The RSSS protocol is an (α, β, δ)-secure PQM protocol for $\alpha < \beta \leq 4\alpha$ as per Definition 1, whenever the probe frequency p and number of packets per interval T satisfy*

$$pT > \ln(\tfrac{1}{\delta}) \tfrac{3}{(\sqrt{\beta}-\sqrt{\alpha})^2} \quad . \tag{5}$$

When $\beta = 2\alpha$ we can use a tighter bound (instead of (5)) to find that when $\delta = 1\%$, we require $pT > 75/\beta$.

When clocks are perfectly synchronized, we omit the proof, since it is almost identical to that of Theorem 2 (because the salt is kept secret until the end of the interval). Furthermore, notice that even dishonest senders cannot bias an honest sender's measurements, since they learn nothing about the salt until the interval is over. Now suppose that Alice's clock lags Bob's clock by at most τ seconds. It follows that there will be period of time of length $< \tau$ where Alice is operating in interval $u-1$ while Bob has already moved into interval u. To deal with this, during the first τ seconds of each 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. While most Internet routers are able to maintain a clock with accuracy of 21ms or less [22], secure clock synchronization is a non-trivial problem. In the full version [14] we show a simple stateless protocol that allows Alice and Bob synchronize their clocks to within 1.5 round trip times.

4.2.2 Transmitting-Server Secure Sampling (TSSS)

We now turn our attention to the case where a single server is *sending* to multiple clients, and each client wants to monitor the traffic it receives from the server while imposing minimal cost on the server. Note that the server is now Alice and the client is Bob. Here the server keeps a single counter per client, and modifies the packets it sends by appending a short MAC tag, that is keyed with *same* key for each client.

The TSSS protocol proceeds as follows. As before, the server picks random salt values $(\mathsf{s}_1(u), \mathsf{s}_2(u))$ at the beginning of the interval, and releases them at the end of the interval. Here, however, the server will keep, for each client B, a count $T_{\mathrm{A}}(B)$ of the number of packets it sends to B during the interval. The server also authenticates all traffic that she sends using the (client-independent) salt: for a packet d, the server will compute a packet digest $z = H(d)$ and then appends the tag $h_{k_2}(u, z)$ to the packet that he sends the client.

The client will randomly sample a p-fraction of the packets received. For each such packet d', he stores the corresponding digests $z' = H(d')$ and the received tag. At the end of the interval, the server reveals the salt as above, and also sends $\mathrm{Sign}_{SK}(T_{\mathrm{A}}(B))$ to B. Each client B verifies the electronic signature and checks all its stored packet digests and tags using this salt. Let T_{B} be the number of valid packets thus found by B; then B estimates the number of failures as $V = pT_A(B) - T_{\mathrm{B}}$. As before, the client raises an alarm if $V > pT_A(B)\sqrt{\alpha\beta}$. Using an argument similar to Theorem 2, the protocol is secure if $\beta < 4\alpha$ and

$$pT_A(B) > \ln(\tfrac{1}{\delta}) \tfrac{3}{(\sqrt{\beta}-\sqrt{\alpha})^2} \quad . \tag{6}$$

4.3 Some sample parameters

Suppose $\beta = 1\%$, and assume a fully utilized 5 Gbps link with an average packet of 3000 bits and an average round trip time (RTT) of 100 msec. Then about $T = 10^7$ packets are sent during an RTT.

Symmetric Secure Sampling. Using the improved bound on Theorem 2 our symmetric sampling protocol is secure when the probe frequency is $p > \frac{75}{\beta T} = 7.5 \times 10^{-4}$. This p is also the communication overhead, *i.e.,* the amount of added ACK packets as a fraction of the data traffic. Using 96-bit packet digests (see Section 3), Alice needs about $pT \sim 90$ KB of storage during a single round trip time. The amount of storage required for Alice can be reduced without compromising security by noting that (2) gives a tradeoff p and T. Alice can decrease her sampling rate to p' if she is willing to use a longer interval $T' = Tp/p'$. Since almost every probe packet tag will be deleted after 1 RTT, this nominally reduces Alice's storage to $p/p' \cdot 90$ KB. This comes at the cost of reduced PQM temporal resolution, due to the longer intervals. (Notice that Alice can arbitrarily decrease her sampling rate without coordinating with Bob simply by changing the parameter p in her Probe function.)

RSSS. As described above, the Receiving-Server Secure Sampling protocol requires the sending client to store information about *every* packet she sends to Bob for the duration of a interval (which may last from a few milliseconds to a few RTTs depending on synchronization quality). In case the intervals last an RTT or more, it is not practical to expect the sender to keep digests of over 10^7 packets in her storage, and so we apply *subsampling* here to reduce the fraction of packets stored: each sender only stores a q fraction of the packets she sent, where each packet is stored independently with probability q. In term of monitoring this is essentially the same as reducing the packet stream by a factor of q, so when $\beta = 2\alpha$ from the improved version of (5) we can see that $pqT > \frac{75}{\beta}$ suffices, giving a tradeoff between storage at Alice qT, and probe frequency and communication overhead p. For example, suppose that the probe frequency is $p = 0.2$. Then, by (5), Alice should store $qT \approx \frac{75}{p\beta} = \frac{75}{0.2 \cdot 0.01} = 3.75 \times 10^4$ packet digests (160 bits each), and about p times as many corresponding ACK tags (96 bits each). Overall, this takes $3.75 \times 10^4 \cdot (160 + 0.2 \cdot 96)/8 \approx 840$ KB of storage. Thus, if intervals last for 1 RTT, so that $T \approx 10^7$, then the subsampling rate must be at least $q = 3.7 \times 10^{-3}$.

TSSS. Here, the sending server stores one 32-bit counter per client, and attaches a 96-bit tag to each message. Following (6), and using same parameters as above, the client needs to store $qT \approx \frac{75}{\beta} = 7.5 \times 10^3$ digests and tags, for a total storage of $7.5 \times 10^3 \cdot (160 + 96)/8 \approx 240$ KB.

5. SECURE SKETCH PQM

In our secure sketch PQM protocol, Alice and Bob aggregate all traffic Alice sends to Bob into a short data structure called a *sketch.* (The difference between a sketch and a sample is that a sketch, although short, depends on all the traffic that was sent/received, rather than just small subset of it.) At the end of the interval, Bob sends his sketch to Alice and she uses the similarity of the sketches to decide whether the failure rate exceeded α.

We can apply several sketching techniques [3, 4, 9, 31] for ℓ_2*-norm estimation* into our framework to give secure PQM protocols. While sketches have been used before in the networking community (to estimate properties of data streams that are too long to be stored in their entirety; *c.f.* [9,31] and the references in [32]), to the best of our knowledge this is the first time that they have been applied to the problem of path-quality monitoring. Furthermore, the special structure in the PQM problem allows us to obtain new and improved analytical bounds on the performance of these schemes. It turns out that the path-quality setting has particular properties that enable us to achieve better performance for some of these schemes. In particular, we prove a new bound on the performance of [9]'s scheme that may be of independent interest.

In this section we start by explaining the relationship between ℓ_2-norm estimation and path-quality monitoring, and then present our PQM protocol and discuss its security. We then show how the protocol works with several known ℓ_2-norm estimation sketches and give settings of parameters based on both analytical guarantees and numerical experiments. Our results show that the secure sketch protocol is almost as lightweight, in terms of storage and communication, as the trivial (but insecure) idea of keeping counters of the number of packets sent and received.

5.1 PQM as norm estimation

We now show how ℓ_2-norm estimation (for which a number of highly efficient and simple schemes are known [**?**, 3, 4, 9]) can be used to realize PQM. Suppose that Alice sends T packets to Bob during some interval. Let U be the total number of all possible packets (*e.g.,* if packets are 1500 bytes long then $|U| = 2^{1500 \cdot 8}$) and let $\mathbf{v}_\mathrm{A}$ be the U-dimensional vector that has c in the position corresponding to packet x if Alice sent x during this interval c times (where c is a non-negative integer). Similarly, let $\mathbf{v}_\mathrm{B}$ be the U-dimensional vector that has c in the x-th position if Bob received packet x exactly c times. Let n_d be the number of packets that were *dropped* during the interval (*i.e.,* sent but not received). Let n_a the packets that were *added* (*i.e.,* received but not sent).

Alice would like to know if more than a β fraction of delivery failures occurred, i.e., whether $n_\mathsf{d} \geq \beta T$. Note that

$$\begin{aligned} n_\mathsf{d} + n_\mathsf{a} &= \sum_x \left| (\mathbf{v}_\mathrm{A})_x - (\mathbf{v}_\mathrm{B})_x \right| \\ &\leq \sum_x \left((\mathbf{v}_\mathrm{A})_x - (\mathbf{v}_\mathrm{B})_x \right)^2 = \left\| \mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B} \right\|_2^2 \end{aligned} \tag{7}$$

where $\|\mathbf{v}\|_2$ denotes the ℓ_2-norm of a vector $\mathbf{v}$ (*i.e.,* , $\|\mathbf{v}\|_2 = \sqrt{\sum_x \mathbf{v}_x^2}$). Furthermore, note that if both $\mathbf{v}_\mathrm{A}$ and $\mathbf{v}_\mathrm{B}$ only have entries in $0, 1$ then $|(\mathbf{v}_\mathrm{A})_x - (\mathbf{v}_\mathrm{B})_x| = \left((\mathbf{v}_\mathrm{A})_x - (\mathbf{v}_\mathrm{B})_x\right)^2$ for every x, and hence the inequality in (7) becomes an equality. Because Alice never transmits duplicate packets (see Section 2), $\mathbf{v}_\mathrm{A}$ is a 0/1 vector, which means that (a) in the benign case where there are no adds and at most αT drops, $\|\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}\|_2^2 \leq \alpha T$ and (b) in the adversarial case, if the adversary drops at least βT packets then $\|\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}\|_2^2 \geq \beta T$, regardless of the number of packet additions. This suggests the following PQM protocol (assuming Bob can transmit his $\mathbf{v}_\mathrm{B}$ securely to Alice): check if $\|\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}\|_2^2 > \alpha T$ and if so, raise an alarm.

Sketches. Obviously, Alice and Bob cannot afford to store or communicate huge vectors $\mathbf{v}_\mathrm{A}$, $\mathbf{v}_\mathrm{B}$, but happily there are known schemes that enable Alice and Bob to maintain short

sketches $\mathbf{w}_A, \mathbf{w}_B$ such that one can estimate $\|\mathbf{v}_A - \mathbf{v}_B\|_2$ from $\mathbf{w}_A$ and $\mathbf{w}_B$ [3,4,9,20,31].[5] Moreover, these sketches can be computed incrementally on a stream of data: we can start with a sketch $\mathbf{w}$ corresponding to the all-zero vector, and for each incoming packet update $\mathbf{w}$ to reflect the increase in one of the coordinates of $\mathbf{v}$; the full vector $\mathbf{v}$ is never stored explicitly. Some differences between the typical usage of these schemes and our setting are:

1. These schemes are probabilistic, typically using a hash function that is known by all parties involved (e.g., a 4-universal hash as in [9,31]). In our setting, however, if the adversary can predict the outputs of the hash function, she can add and drop packets in a way that cannot be detected (*e.g.,* by dropping some packet and replacing it with a different packet that maps to the sketch in an identical way). For this reason, we replace the public hash function with a *keyed* hash function, with a secret key is shared between Alice and Bob, and is refreshed every interval.

2. Because we only need to detect when the failure rate is above a certain threshold, we can use a far coarser estimation than the typical applications of such sketches, and can choose parameters for these sketches that result in very little storage and communication.

3. The fact that Alice does not send duplicate packets during an interval implies a special structure of the vectors $\mathbf{v}_A, \mathbf{v}_B$ that allowed us to even further improve the parameters of the sketches, see Theorem 5.

5.2 The secure sketch protocol

Recall that the inputs to a PQM protocol are the thresholds α, β such that it should raise an alarm if more than a β fraction of the packets are tampered with and not raise an alarm if less than an α fraction are dropped. Our protocol works in separate intervals. We assume Alice and Bob share a secret (master) key k, and derive an interval key k_u for each interval u (see Section 3). In the full version [14], we provide a detailed treatment of techniques that Alice and Bob can use to synchronize their intervals; below we assume that Alice and Bob agree on an interval of T packets. Within interval u, our secure sketch protocol operates as follows:

1. Alice runs a sketching algorithm, using the PRF $h_{k_u}(\cdot)$ as the hash function, to incrementally compute a sketch $\mathbf{w}_A$ of the vector $\mathbf{v}_A$ induced by the packet it sends. (Since we using sketches that are computed for streaming data, this amounts to running an update algorithm that maps each sent packet to the sketch.) Alice and Bob use shared secret randomness k_u for this algorithm, and k_u is refreshed using (a PRF keyed with) the master key at every interval.

2. Bob similarly uses $h_{k_u}(\cdot)$ to compute sketch $\mathbf{w}_B$ of the vector $\mathbf{v}_B$ induced by the packets he receives.

3. At the end of the interval, Bob sends his sketch $\mathbf{w}_B$ to Alice, labeled with interval number u and authenticated using a MAC.

4. Alice computes an estimate V of $\|\mathbf{v}_A - \mathbf{v}_B\|_2^2$ and raises an alarm if and only if $V > \frac{2\alpha\beta}{\alpha+\beta}$.

The decision threshold $\Gamma = 2\alpha\beta T/(\beta+\alpha)$ is used by Alice to decide between cases where $n_d < \alpha T$ and $n_d > \beta T$. (We derived Γ using maximum likelihood estimation, under the assumption that V is distributed like a Gaussian random variable. Later we show that this threshold works well even though V is not exactly Gaussian.)

Note that our protocol does not require Bob to send Alice his sketch immediately after the interval ends; sketches from multiple intervals can be grouped together and sent at the end of a set of intervals (this only effects the delay before Alice can decide if she should raise alarm).

THEOREM 4. *Suppose that the sketch guarantees, when using a truly random hash function, that with probability at least $1-\delta$, the estimate of square norm $\|\mathbf{v}\|_2^2$ is within $(1\pm\epsilon)$ for $\epsilon = \frac{\beta-\alpha}{\beta+\alpha}$. Then, the secure sketch protocol is a (α,β,δ)-secure PQM protocol as per Definition 1.*

PROOF OF THEOREM 4. First observe that Eve cannot forge the report that the Bob sends to Alice, since the report is authenticated using a secure MAC (and dropping the report will only cause Alice to raise an alarm). It follows that at the end of the interval Alice gets a consistent version of Bob's sketch $\mathbf{w}_B$. Now, the derived estimate V is as good as if the sketch used a truly random hash function; indeed, if Eve could bias V then the pseudorandomness of the secretly-keyed PRF $h_{k_u}(\cdot)$ would be violated (note that no effect of this PRF is visible to Eve until the end of the interval using k_u). Thus, then our assumption about the sketch scheme guarantees that with probability $1-\delta$:

1. No false positives: if $n_d \le \alpha T$ and $n_a = 0$, then $\|\mathbf{v}_A - \mathbf{v}_B\|_2^2 \le \alpha T$ and the estimate V is at most $(1+\frac{\beta-\alpha}{\beta+\alpha})\alpha T = \frac{2\beta\alpha}{\beta+\alpha}T = \Gamma$.

2. No false negatives: if $n_d > \beta T$, then $\|\mathbf{v}_A - \mathbf{v}_B\|_2^2 > \beta T$ and the estimate V is greater than is $(1-\frac{\beta-\alpha}{\beta+\alpha})\beta T = \frac{2\beta\alpha}{\beta+\alpha}T = \Gamma$.

1. and 2. guarantee that with high probability Alice can use the decision threshold Γ to decide between cases where $n_d < \alpha T$ and $n_d > \beta T$. □

From the proof, we see that it suffices if the sketch guarantees that the estimate is at most $(1+\epsilon)\alpha T$ for vectors $\mathbf{v}$ that have all entries in $\{-1,0,1\}$ and with norm $\|\mathbf{v}\|_2^2 \le \alpha T$, and the estimate is at least $(1-\epsilon)r$ for vectors $\mathbf{v}$ that have at least $r \ge \beta T$ entries in ± 1 (and possibly other nonzero entries as well). It turns out this observation is crucial for obtaining improved parameters for our protocol; see Theorem 5 below.

Hashing every packet. Notice that the PRF that is used to hash packets into the sketch has a small output range (*i.e., N* instead than a exponentially large space). This suffices because we only require that an adversary cannot predict the bin that a packet hashes to with probability greater than $\frac{1}{N}$. Furthermore, because no information about the bin a packet hashes to leaks out until the end of the interval (when the report is sent) and parties refresh their keys for each new interval, the protocol remains secure even if a faster pairwise independent hash (*e.g.,* [8]) is used instead of a PRF. (However, using this weaker hash function comes at the cost of worse sketch parameters N; in particular, Theorem 5 below no longer holds.)

[5] Estimating the ℓ_p-norm for any $p \ge 1$ would also suffice to satisfy the analog of (7), and indeed such sketch schemes exist. However, the presently known schemes for ℓ_2-norm estimation are more efficient.

5.3 Plugging in ℓ_2-norm estimation schemes

In this section we show how to instantiate our PQM protocol with known sketching schemes, such that the sketching schemes satisfy the requirements of Theorem 4. We start with the classic sketching technique [3, 4] based on the Johnson-Lindenstrauss lemma, and then show how the sketch of Charikar, Chen and Farach-Colton [9] can improve performance. We show how these schemes compare in terms of update time per incoming packet and storage requirements (*i.e.,* the number of bins in the sketch, N, and the size of each bin). We also derive new bounds for the storage requirements of these schemes for our setting.

All of the ℓ_2-norm estimation schemes we consider have the following form. They transform a U-dimensional vector $\mathbf{v}$ into a shorter, N-dimensional vector $\mathbf{w}$ by choosing a random linear map S from some set $\mathcal{S}$ and setting $\mathbf{w} = S(\mathbf{v})$. Then an estimator V for $\|\mathbf{v}\|_2^2$ is computed from $\mathbf{w}$; in the cases we consider, the estimator will simply be $\|\mathbf{w}\|_2^2$.

Due to linearity, we can compute $\mathbf{w}$ using streaming access to the vector $\mathbf{v}$. That is, we initialize $\mathbf{w}$ to be all zeroes, and then when we see a packet x we can update the sketch as $\mathbf{w} \leftarrow \mathbf{w} + S(\mathbf{e}_x)$, where $\mathbf{e}_x$ is the U-dimensional vector with 1 in the x^{th} position and 0 everywhere else. In general this requires updating all N positions of $\mathbf{w}$, but if $S(e_x)$ is, say, zero everywhere except for one entry for every x, then we only need to make a single update to $\mathbf{w}$. Again, linearity implies that $\mathbf{w}_\mathrm{A} = S(\mathbf{v}_\mathrm{A})$ and $\mathbf{w}_\mathrm{B} = S(\mathbf{v}_\mathrm{B})$, we have $\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B} = S(\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B})$. Hence if Alice and Bob want to compute the distance of $\mathbf{v}_\mathrm{A}$ and $\mathbf{v}_\mathrm{B}$, they can do so by using the same function S to compute the respective sketches $\mathbf{w}_\mathrm{A}$ and $\mathbf{w}_\mathrm{B}$. Then Bob sends $\mathbf{w}_\mathrm{B}$ to Alice and she runs the estimator on the difference vector $\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B}$. All of the schemes we consider are known to have estimators V with expectation $\|\mathbf{v}\|_2^2$ and variance $\frac{2}{N-1}\left(\|\mathbf{v}\|_2^4 - \sum_x (\mathbf{v}_x)^4\right)$.

Classic dimension-reduction sketches. The original sketch of Johnson and Lindenstrauss chose S to be a projection into a random N-dimensional hyperplane. Indyk and Motwani [19] showed that S can be a random $N \times U$ matrix whose entries are independent Gaussian random variables with mean 0 and variance $1/\sqrt{N}$, and Achlioptas [3] (see also [4]) showed that the entries can simply be chosen as either $\frac{+1}{\sqrt{N}}$ or $\frac{-1}{\sqrt{N}}$ with probability 1/2 each. In all of these cases, to ensure that with probability $1-\delta$ the square norm of $\mathbf{w} = S(\mathbf{v})$ is within a $(1 \pm \epsilon)$ factor of $\|\mathbf{v}\|_2^2$, we can take $N = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$; specifically Achlioptas [3] showed that it suffices to choose

$$N > \frac{12}{\epsilon^2}\frac{1}{3-2\epsilon}\ln\frac{1}{\delta} \tag{8}$$

To use this scheme in our context, when receiving a packet d we use a hash function that maps it to a vector $b \in \{\pm 1\}^N$ and add b to the sketch $\mathbf{w}$. To prevent overflow we can take each bin in the sketch to hold numbers in $[-K, +K]$ where $K = 2\sqrt{\log(4N/\delta)T/N}$. (We can change the protocol to raise an alarm if any bin overflows, since this will happen with low probability in the benign case.)

CCF sketch. Charikar, Chen, and Farach-Colton [9] gave a scheme with a faster update time; instead of updating *all* N bins each time a new packet arrives, the CCF scheme only updates a single bin. CCF uniformly draws S from $\mathcal{S}_\mathrm{CCF}$, the set of random sparse $N \times U$ matrices in which every column is all-zero except for a single entry which is ± 1. In our context, this means that when receiving a packet d we use a hash function that maps it to a pair (i, b) where $i \in [N]$ and $b \in \{\pm 1\}$, and add b to the i^{th} bin in the sketch $\mathbf{w}$. To prevent overflow we can again take each bin to hold numbers in $[-K, +K]$ where $K = 2\sqrt{\log(4N/\delta)T/N}$.

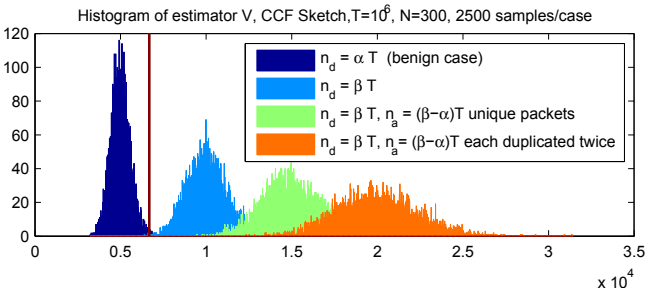

Figure 2: Histogram of estimator for the CCF schemes with $N = 300$, $T = 10^6$, $\alpha = \frac{1}{2}\beta$, $\beta = 1\%$ and threshold $\Gamma = 6667$.

In order to get a $(1 \pm \epsilon)$ accuracy with probability $1 - \delta$ this schemes require a larger N, $N = \Theta(\frac{1}{\delta\epsilon^2})$, rather than $N = \Theta\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ of the classic scheme; it turns out no sparse scheme (that updates only a single bin for each incoming item) can achieve a better bound when the input can be arbitrary.[6] Thus, in general there is an inherent tradeoff between storage size and update speed. Fortunately, it turns out that CCF performs well in our setting because Alice sends unique packets, and so our vectors are not sparse. In fact, we prove below that in our setting it *does* suffice to use $N = O(\frac{\log(1/\delta)}{\epsilon^2})$. Our improved bound relies both on the fact that our vectors $\mathbf{v}$ have many non-zero entries and the fact that we care about deciding whether $\|\mathbf{v}\|_2^2$ lies above or below a threshold rather than getting an accurate estimate.

THEOREM 5. *For every $\delta, \epsilon > 0$, let $\mathbf{v}$ be a U-dimensional vector all of whose entries are in $\{-1, 0, 1\}$ and $\mathbf{w} = S\mathbf{v}$ where S is an $N \times U$ matrix chosen uniformly at random from the set $\mathcal{S}_\mathrm{CCF}$. If*

$$\|\mathbf{v}\|_2^2 > q = \tfrac{6}{\epsilon^2}N(\ln N + \ln\tfrac{2}{\delta}) \tag{9}$$

$$N > \tfrac{24}{\epsilon^2}(1+\epsilon)^2\ln\tfrac{2}{\delta} \tag{10}$$

then, with probability $1 - \delta$, we have $\|\mathbf{w}\|_2^2 \in (1 \pm \epsilon)\|\mathbf{v}\|_2^2$.

See the full version [14] for a tighter and more precise statement of Theorem 5, as well as its proof.

To apply the theorem into our setting, set $\epsilon = \frac{\beta-\alpha}{\beta+\alpha}$, set $\mathbf{v} = \mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}$, and set $q = \alpha T$. The false positive condition is satisfied because we have $\mathbf{v} \in \{0, 1\}^U$ and $\|\mathbf{v}\|^2 \le \alpha T$, so with probability $1 - \delta$, $V = \|\mathbf{w}\|^2 < \frac{2\alpha\beta T}{\alpha+\beta}$. The false negative condition is satisfied because we have the number of drops is $r > \beta T$ (and packets dropped are unique so they

[6] CCF's [9] sketch can attain better success probability by using the median of estimates obtained from M independent sketches, for some number M. However, this increases the storage and update time by a factor of M. Also note that [9] can show, using 4-universal hashing and Chebyshev's inequality, that the number of bins relates to the error probability as $N = O(\frac{1}{\delta})$. However, even if the hash is a completely random function, it is impossible to get a better error probability for general vectors (this is shown in the full version of this paper).

each correspond to a 1 entry in $\mathbf{v}$), and so, with probability $1-\delta$, we get that $V = \|\mathbf{w}\|^2 > \frac{2\alpha\beta T}{\alpha+\beta}$.

Notice the bound requires conditions on both $\|\mathbf{v}\|_2^2$ and N. The fact that N, the number of bins in the sketch, must be large is not so surprising. We need $\|\mathbf{v}\|_2^2$ to be large because CCF does not work as well when very sparse vectors $\mathbf{v}$ cause high variance in the number of entries in the bins of $\mathbf{w}$. This condition on $\mathbf{v}$ holds in our setting because the number of bins in the sketch is much smaller than the total number of packets. Similar conditions apply in many other sketch applications, and thus this theorem may be of independent interest.

TZ Sketch. Thorup and Zhang [31] gave a variant of the CCF scheme where, instead of updating a bin in the sketch with a randomly chosen element in $\{\pm 1\}$, the bin is always updated with a $+1$. However, their scheme requires a larger bin size (roughly twice the number of bits/bin) than CCF.

Adversarial sketch model. Mironov et.al. [23] recently found schemes that allow Alice and Bob to sketch *adversarially-chosen* sets without using any shared randomness (*i.e.,* keys), and then use a secure channel to exchange sketches. Their result is useful for the client-server setting, where a server wants to engage in a sketching protocol with multiple clients without using a shared key for each, and then uses a report authenticated using a public key, *c.f.,* Section 4.2. However, an adversarial-sketch protocol that does *not* use shared randomness (keys) to create the sketch, needs at least $O(\sqrt{T})$ storage [23], and is thus much less efficient than $O(\log T)$-storage *keyed* sketches that we considered here.

5.4 Some sample parameters and experiments

Suppose the detection threshold is $\beta = 0.01$, the false alarm threshold is $\alpha = \beta/2$ and that about $T = 10^7$ packets are sent during an interval. Then, if we want a confidence of $1-\delta = 99\%$, we can use (8) to find that PQM protocol based on the classic scheme requires $N = 214$ bins with $b = 14$ bits per bin (for an array size of ~ 525 bytes). For the CCF scheme, we can apply the refined version of Theorem 5 in the full version [14] for the same α, β, δ, to find that we can use $N = 300$ counters of $b = 14$ bits if we take intervals containing at least $T = 10^9$ packets. As we discuss below, our numerical experiments suggest that even better parameters are achievable for the CCF protocol. They indicate (though do not conclusively prove) that even for $T = 10^7$ we can use $N = 150$ bins with 14 bits per bin, to get a total sketch size of roughly 200 bytes. Indeed, CCF seems like the best of the schemes we considered, since it has a faster update time than the classic scheme (and less bits per bin than TZ).

Figure 2 is a histogram of CCF estimators V for (from left to right) the benign case where $n_d = \alpha T$ (here we want the estimator to be below the threshold Γ so that Alice does not raise an alarm), and for three cases where $n_d = \beta T$ so we want Alice to raise an alarm: a case where Eve does not add any packets, a case where Eve adds $(\beta-\alpha)T$ distinct packets, and a case where Eve adds $(\beta-\alpha)T$ total packets where each packet is duplicated twice. Notice that the threshold Γ clearly distinguishes between cases where $n_d = \beta T$ and the benign cases where $n_d = \alpha T$. Also, notice if Eve adds packets to the link, she only increases the probability that Alice raises an alarm, as predicted by (7). (Indeed, in the full version [14] we prove a theorem that gives evidence that Eve cannot improve her chances of tricking Alice into not raising an alarm when $n_d > \beta T$ by adding packets to the data path.) Figure 2 suggests that taking $N = 300$ suffices for CCF even if we have shorter interval lengths of $T = 10^6$. In Figure 3 we show the probability that Alice raises an alarm vs the number of bins N in the CCF sketch, in each of the four cases we described above. Even when we consider short interval lengths of $T = 10^5$ our experiments suggest that using a CCF array with $N = 150$ bins suffices for a statistical confidence of $\delta = 1\%$.

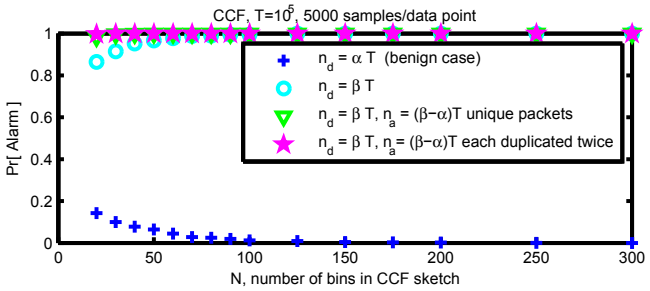

Figure 3: Alarm probability versus N, the number of bins in the CCF sketch. $\alpha = \frac{1}{2}\beta$, $\beta = 1\%$, $T = 10^5$ **packets per interval and threshold** $\Gamma = 666.7$.

6. NECESSITY OF CRYPTOGRAPHY

All of our protocols require keys between participating nodes, and cryptographic computations. We now show that this overhead is inherent by arguing that *any* PQM protocol satisfying Definition 1 requires a key infrastructure and the invocation of cryptographic operations. These results also immediately imply that any PQM protocol that does not use keys or cryptography, *e.g.,* Listen [30], is insecure according to Definition 1.

Keys are necessary. To see this, we argue in the contrapositive: suppose Bob has no secrets from Eve. Then, since Eve occupies a node on the path between Alice and Bob, she receives the same information that Bob receives and can compute the same responses. It follows that Eve can simply run the PQM protocol on her own (responding to Alice with the appropriate acks or reports), and then drop all the packets going to Bob. This breaks security because Alice has no way to know that anything went wrong. Notice further that this suggests that Alice needs Bob's participation in order to run a secure PQM protocol.

Cryptography is necessary: We now argue that the keys must be used in a "cryptographically-strong" manner. Note that our previous result that keys are necessary does not imply that cryptography is necessary; for example [12] uses secret keys in a non-cryptographic way and obtains a protocol that is not secure by our definitions. To show that cryptography is necessary, we show that any secure PQM protocol is at least as complex as a secure *keyed identification scheme (KIS)*, which is known to be equivalent to many cryptographic tasks like encryption and message authentication [18]. Intuitively, our result follows from the fact that in order for Alice to believe Bob, she must be assured that all the information she is getting indeed came from Bob in a way that Eve cannot impersonate. Our reduction can be found in the full version [14].

7. COMPARISON OF PROTOCOLS

Because we want PQM protocols that can be deployed in high-speed routers, we have focused on efficiency considerations; namely, we evaluated our protocols' efficiency in (a) *communication overhead*, (b) *computation* of cryptographic operations, and (c) use of dedicated *storage* in the router. We now explore a wider space of design objectives for evaluating our PQM protocols, discuss how our three protocols perform under these objectives, and compare them with two existing solutions for PQM: Stealth Probing [5] and IPsec. We argue that obtaining PQM protocols that perform well for one particular objective often involves trading off some other objective. This discussion is summarized in Table 1.

Secure sketching. Our secure sketch protocol makes extremely efficient use of storage and communication. Furthermore, these requirements are (roughly) independent of the threshold chosen, and so can be used even to detect very small degradations in path performance. On the other hand, the secure sketch protocol does not allow us to easily measure round trip time, since packets are aggregated into one sketch. It requires both the sender and the receiver to maintain keys and (small) storage, which might be a problem in the client/server setting where a server is communicating with many clients, and does not want to maintain per-client storage for the purposes of running PQM protocols. Finally, the sketch protocol is not *monotone*: it will raise an alarm if many packets are *added* into the path, even if no packet is actually dropped. This could be an issue if an adversary that does *not* sit on the path is able to inject packets into the path.

Secure sampling. Our secure sampling protocols are best suited for situations where Alice needs immediate feedback and accurate measurements of round-trip delay (which she can easily obtain, even in the absence of synchronized clocks, by timing the arrival of acks). Furthermore, the protocols are *monotone* in the sense that if an adversary adds packets to the path or spoofs acks, Alice can simply ignore all the acks that do not correspond to the packets that she sent. *Symmetric Secure Sampling* is best suited when Alice and Bob are peers that have equal resources to devote to the protocol. Furthermore, the protocol is best when we do not want to make any clock synchronization assumptions, or when we want fast feedback (which can be obtained by adjusting the probe frequency p appropriately, see Section 4.3). *Asymmetric Secure Sampling* is best suited for the client-server setting, where the server wants to run PQM protocols with many clients without using dedicated storage and using only a single key for all clients.

However, the sampling protocols (save for the TSSS protocol of Section 4.2.1) have a disadvantage in the *asymmetric path* setting— when the forward (Alice to Bob) path is not the same as the reverse (Bob to Alice) path. The reason is that since only a p-fraction of sent packets are acknowledged, each dropped ack looks like $\frac{1}{p}$ dropped packets. Thus, in the asymmetric path setting, an adversary on the reverse path can arbitrarily increase the source's estimate of the failure rate on the forward path by dropping acks. In contrast, in the secure sketch protocol only a single authenticated report packet is sent on the reverse path, and so if it does not arrive Alice can deduce that the problem is in the reverse rather than the forward path (unless the forward path is completely blocked and Bob is not even aware of Alice's existence). This issue also means that the sketch protocol is better suited for SLA-compliance monitoring applications, especially in the asymmetric paths setting (where the report packet could even be sent out-of-band). When PQM is used to inform routing decisions in the asymmetric setting, Alice and Bob can always coordinate switching their forward and reverse paths once an alarm is raised.

	Secure Sampling		Secure
	Sym	Asym	Sketching
Storage/Communication[7]	90KB	240–840KB	0.2–0.6KB
Peer setting	✓		✓
Client-server setting		✓	
No clock sync	✓	coarse	✓
Estimates delay	✓	✓	
Fast feedback	✓		
Monotonicity	✓	✓	

Table 1: Tradeoff space for our protocols.

IPsec. IPsec is a standard for symmetric-key encryption and authentication of packets at the network layer. However, it requires invoking a cryptographic operation, modifying, and adding tags to every packet sent on the path, which could be quite expensive when operating at multi Gbit/sec rates. Also, IPsec currently does not include a standard for providing authenticated acknowledgments and so needs additional machinery, like *Stealth Probing* [5], in order to provide secure PQM at the network layer. On the other hand, if we perform PQM at a higher layer, we can use TCP over IPsec (so that we have authenticated acknowledgments for every single packet sent) or even SSL. These protocols provide very strong security guarantees; they not only provide confidentiality, but also allow a source to detect if a failure occurs for *every single packet it sends*. But given the high cost associated with these guarantees, these protocols are arguably, more appropriate when confidentiality and integrity are necessary for other reasons, or when PQM functionality is required at the end-host, rather than in the high-speed routing setting that we focus on here.

Note that all our protocols can be tuned to measure the performance on a particular subset of the traffic, for the purposes of detecting whether some intermediate nodes treat certain packets (such as Skype [25] or BitTorrent [1]) differently than others. The same is true for IPsec based solutions such as Stealth probing. In fact, the latter solutions make selective (mis)treatment of packets by the adversary much harder, as they encrypt all traffic.

8. CONCLUSION

In this paper, we have designed and analyzed efficient path-quality monitoring protocols that give accurate estimates of path quality in a challenging environment where adversaries may drop, delay, modify, or inject packets. Our protocols have reasonable overhead, even when compared to previous solutions designed for the *non-adversarial* settings, and all except TSSS do not modify data packets in any way. In fact, one possible deployment scenario for our protocols is to start by deploying protocols that use hash functions with publicly-known keys, to monitor path quality in manner that is robust to non-adversarial failures such as congestion, misconfiguration, and malfunctions. Then, the same router support could be leveraged, using secret keys,

[7]Storage and communication are given for an interval of $T = 10^7$ packets with $\beta = 0.01$, $\alpha = \beta/2$, and $1 - \delta = 99\%$.

to operate in an adversarial setting as needed. Another possibility is to use our protocols with publicly known keys, but combine them with IPsec for paths where protection against adversarial nodes is required; this will be secure, albeit at a higher overhead than using our protocols on their own.

In our ongoing work, we are investigating the target applications of our protocols: driving routing decisions and detecting violations of Service-Level Agreements. Accurate techniques for determining when performance degrades beyond a threshold will offer significant improvements for edge networks balancing load over multiple paths through the Internet. In addition, we are exploring how to *compose* multiple instances of our PQM protocols—running over multiple paths simultaneously—to determine whether the adversary resides on either the forward or reverse path, or to localize the adversary to particular nodes or links [7]. We believe that our PQM protocols, and our associated models of their properties, are valuable building blocks for designing future networks with predictable security and performance.

Acknowledgments. The authors thank Eugene Brevdo, Moses Charikar, Nick Feamster, Piotr Indyk, Changhoon Kim, Amir Shpilka, and Yi Wang for useful discussions, and Elliott Karpilovsky, Haakon Ringberg, Augustin Soule and the anonymous SIGMETRICS reviewers for comments that have greatly improved the presentation of this work.

S.G. and J.R were supported by HSARPA grant 1756303. D.X. was supported by an NDSEG Graduate Fellowship and an NSF Graduate Research Fellowship. E.T. was supported by Rothschild Fellowship. B.B. was supported by NSF grants CNS-0627526 and CCF-0426582, US-Israel BSF grant 2004288 and Packard and Sloan fellowships.

9. REFERENCES

[1] Bad ISPs that cause trouble for BitTorrent clients. `http://www.azureuswiki.com/index.php/Bad_ISPs`.

[2] Keynote launches new SLA services, June 2001. `http://investor.keynote.com/phoenix.zhtml?c=78522&p=irol-newsArticle_Print&ID=183745`.

[3] D. Achlioptas. Database-friendly random projections. In *PODS*, pages 274–281, 2001.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.

[5] I. Avramopoulos and J. Rexford. Stealth probing: Data-plane security for IP routing. *USENIX*, 2006.

[6] H. Ballani, P. Francis, and X. Zhang. A study of prefix hijacking and interception in the Internet. In *ACM SIGCOMM*, 2007.

[7] B. Barak, S. Goldberg, and D. Xiao. Protocols and lower bounds for failure localization in the Internet. In *IACR EUROCRYPT*, 2008.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *JCSS*, 18(2):143–154, 1979.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

[10] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), 2006.

[11] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *J. of the ACM*, 40(1), 1993.

[12] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Networking*, 9(3), 2001.

[13] S. Goldberg and J. Rexford. Security vulnerabilities and solutions for packet sampling. *IEEE Sarnoff Symposium*, 2007.

[14] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-quality monitoring in the presence of adversaries. Technical report, Princeton University Department of Computer Science, 2008.

[15] K. J. Houle and G. M. Weaver. Trends in denial of service attack technology. Technical report, CERT Coordination Center, 2001.

[16] IETF. Packet sampling working group. *http://www.ietf.org/html.charters/psamp-charter.html*.

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

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

[19] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[20] W. Johnson and J. Lindenstrauss. Extensions of Lipshitz mapping into Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[21] M. Luckie, K. Cho, and B. Owens. Inferring and debugging path MTU discovery failures. In *Internet Measurement Conference*, 2005.

[22] D. Mills, A. Thyagarajan, and B. Huffman. Internet timekeeping around the globe. *Proc. PTTI*, pages 365–371, 1997.

[23] I. Mironov, M. Naor, and G. Segev. Sketching in adversarial environments. In *STOC*, 2008.

[24] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Detecting and isolating malicious routers. *IEEE Transactions on Dependable and Secure Computing*, 3(3):230–244, 2006.

[25] A. Nucci. Skype detection: Traffic classification in the dark, 2006. `http://www.narus.com/_pdf/news/Converge-Skype%20Detection.pdf`.

[26] 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.

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

[28] J. Sommers, P. Barford, N. Duffield, and A. Ron. Accurate and efficient SLA compliance monitoring. In *ACM SIGCOMM*, 2007.

[29] J. Stone and C. Partridge. When the CRC and TCP checksum disagree. In *ACM SIGCOMM*, 2000.

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

[31] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *SODA*, pages 615–624, 2004.

[32] J. Xu. Tutorial on network data streaming. In *ACM SIGMETRICS*, 2008.