# Path-Quality Monitoring in the Presence of Adversaries

.

(This is the full version of [21] from June 12, 2009)

Sharon Goldberg*, David Xiao*, Eran Tromer†, Boaz Barak*, and Jennifer Rexford*
*Princeton University, †MIT

## 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.

This is an extended and revised version of the paper [21] "Path Quality Monitoring in the Presence of Adversaries" that appeared in the Proceedings of ACM SIGMETRICS 2008.

## 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.

### 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 [8], or compromise one of the routers along an existing path through the Internet [23, 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 [37] 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 [31]. As another example, link-level CRC checks are surprisingly ineffective at detecting the kinds of errors that corrupt IP packets [46]. 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 [19, 24] 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 $\beta$ (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 [47] 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 $\ell_2$*-norm estimation sketches* [3,5,13,48] 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 [13]'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 $\beta$, 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 2.1) 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 Appendix A). 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. Fix a set of $T$ consecutive packets sent by Alice, 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 last packet of interval arrives at Bob). 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 2.1. Given parameters $0 < \alpha < \beta < 1$ and $0 < \delta < 1$, we say a protocol is a $(\alpha, \beta, \delta)$ *secure PQM protocol* if, letting $T$ be the number of packets sent during the interval:

1. *(Few false negatives.)* If more than $\beta T$ 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 $\alpha T$ failures occur then the protocol raises an alarm with probability at most $\delta$.

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 [19] (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 $\beta$. 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 more 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.

Next, while our security definition requires that our protocols do not raise a (false) alarm when the one-way failure rate is less than $\alpha$ for the *benign setting*, we do allow for the possibility of raising an alarm due to adversarial tampering even when fewer than an $\alpha$ 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.

Finally, we note that in principle, $\alpha, \beta$ can be chosen arbitrarily, there are a number of practical issues involved in the choice of these parameters. Firstly, we shall show in Sections 4-5 that the (communication, and storage) overhead of our protocols is related to the ratio $\frac{\alpha}{\beta}$; a smaller ratio leads to less overhead. Furthermore, the absolute value of $\alpha$ is sometimes constrained by interval synchronization; we discuss these issues further in Appendix B.

## 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, [25,43,44] 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 [25] and Orchid [36] or implicitly as in Trajectory Sampling [19] and PSAMP [24], 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 [20] 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 [19,24], that are designed for characterizing traffic mix. Similarly, our secure sketch protocol draws on $\ell_2$-norm estimation schemes [3, 5, 13, 48] that are typically uses to characterize data streams.

(See *e.g.,* [50] 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 5.4 we discuss how the new result of [34] 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.,* [18] [39] and later works. While *statistical* PQM protocols have been considered in the security literature [7, 35, 47], 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 [35] PQM approaches is based on a simple counter (and is therefore vulnerable to the attack described above), while Listen [47] 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 [7] 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:

**Keys.** Each of our protocols require some sort of key infrastructure; the secure sketch (Section 5) and symmetric secure sampling (Section 4.1) protocols require parties to share a pairwise secret key, while the asymmetric secure sampling protocol (Section 4.2) require public-keys. Notice that the requirement for pairwise secret keys, 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 [17]) 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. Our protocols require two types of keys: *master keys*, and *interval keys*. Master keys are strong keys that are set up when the protocol initializes, and must remain secure for the lifetime of the protocol. Interval keys are ephemeral keys that are derived at the beginning of each interval, and must remain secret only while packets belonging to that interval are in flight on the path between Alice and Bob.

**Collision-Resistant Hash (CRH) function** is a function $H$ for which it is infeasible (for any computationally-bounded algorithm) to find a collision, *i.e.,* $m \neq m'$ fulfilling $H(m) = H(m')$. (This informal definition suffices for the purposes of this paper. For a more precise definition of CRH see [41].) 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.

**PseudoRandom Function (PRF)** [22] 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 usually 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 insignificant 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. Our protocols use PRFs in two ways.

- A PRF $h$ is used to derive interval keys from the pairwise shared master key and the interval number. To derive interval key $k_u$ for interval $u$ from master key $k$, each party need only compute $k_u = h_k(u)$. Notice that as long as parties have synchronized their interval numbers $u$, they can use their knowledge of the master key $k$ to independently compute $k_u$ (without requiring a key-agreement protocol or a handshake).

  Because the PRF $h$ is used only once per interval, and also needs to be resilient against many queries, we will let $h$ be traditional conservative pseudorandom function. The most common way to (heuristically) realize pseudorandom hash functions (PRFs) is using a full-fledged cryptographic hash functions such as SHA-1 in HMAC mode [28], or with a block cipher like AES in a MAC mode of operation. Their typical performance in a software implementation is 10–20 cycles per input byte, which suffices for many applications.

- All our protocols require a hash computation on the entire contents of every sent packet,[1] and all subsequent processing of the packet relies only on this hash value. For packet hashing, we will use a PRF keyed with the interval key $k_u$. The $k_u$ is used only for the duration of single interval (typically about 100ms); once the interval ends, the key no longer needs to be kept secret. It follow the security requirement on this PRF is weaker than is typically required for most applications. Thus, our packet-hashing PRF should be (a) fast enough to keep up with multi-Gbit/sec packet streams, (b) remain secure after $T = 10^7$ applications and/or for about 100ms. While designing PRFs that are especially suited to this purpose remains an interesting area for future research, in Appendix A we discuss some realizations of our packet-hashing PRF in *both hardware and software* based on known cryptographic hash functions.

**Universal hash functions** are keyed hash functions similar to PRFs, but have a weaker security requirement; PRFs are indistinguishable from functions that map *every* input to an random *independent* outputs, while universal hash functions only require independence between some small number of outputs [12]. In Sections 5.4 and 5.4.4, we shall show

[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 [19, Section II.A]).

that packet hashing can sometimes be performed using these weaker hash functions, instead of PRFs.

In this work, we consider two types of universal hash functions.

- $\varepsilon_g$-**almost universal hash function** $g$ producing $n$-bit outputs guarantees that for any pair of distinct inputs $x, x'$, then

$$\Pr\left[g_{k_u}(x) = g_{k_u}(x')\right] \leq \varepsilon \quad (1)$$

  where the probability is over the choice of $k_u$ used to key $g$. There are many possible realizations of such hash functions, see for example [10] for a survey. In this work, we sometimes use GHASH [32] to compute sample parameters for our constructions. GHASH is an $\varepsilon_g$-almost universal hash function that produces $n$ bit outputs. GHASH hashes variable-length packets by breaking the packet into blocks of length $m$ and iteratively hashing each block. For a packet of length $\ell$, GHASH has $\varepsilon_g = \frac{\ell}{m} 2^{-n}$.

- **4-wise independent hash function** $g$ producing $n$-bit outputs guarantees that for any four distinct inputs $x_1, x_2, x_3, x_4$ and (not necessarily) distinct outputs $y_1, y_2, y_3, y_4$, then

$$\Pr\left[g_{k_u}(x_i) = y, \forall i = 1...4\right] = \left(\tfrac{1}{2^n}\right)^4 \quad (2)$$

  where the probability is over the choice of $k_u$ used to key $g$. We shall use 4-wise independent hash functions to realize (theoretically)-faster packet hashing in our 'secure sketch' protocol. To realize a 4-wise independent hash function, we use polynomials of degree 3 [12], for example, to compute $g_{k_u}(x)$ set key $k_u = (a_0, a_1, a_2, a_3)$ and output $a_3x^3 + a_2x^2 + a_1x + a_0$. This computation can be done in three multiplications using Horner's rule.

Interestingly, in practice, a PRF could be faster than a 4-wise independent function (even though a PRF provides strictly stronger theoretical guarantees). This is because we typically realize 4-wise independent functions based on constructions that come with rigorous proofs of correctness *e.g.,* polynomials of degree 3 [12]. Meanwhile, in practice we use PRFs that come with only *heuristic* guarantees of correctness, and can thus faster (*e.g.,* GHASH-AES [32] is a PRF under the heuristic assumption that AES is a fixed-input-length PRF, see Appendix A.) However, even fast (heuristic) constructions of PRFs are typically based on $\varepsilon_g$-almost universal hash functions (see Appendix A), and as such, we can safely assume that $\varepsilon_g$-almost universal hash functions are always faster than PRFs.

**Message Authentication Code (MAC)** is a basic cryptographic primitive that can be realized using a PRF: 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 $\text{MAC}_k(m) = (m, h_k(m))$. We shall assume that the MAC tag (*i.e.,* PRF output) is $n = 96$ bits.

**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 $\sigma$; we denote this with $\sigma = \text{Sign}_{SK}(m)$. A public key $PK$ is known to all parties and is used to verify the signature; the $\text{Verify}_{PK}(\sigma)$ 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.

# 4. SECURE SAMPLING PQM

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 Appendix C 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 receivers) use a variant of *delayed-exposure techniques* (*c.f.,* TESLA [40] [9, 11, 14] 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. These protocols also have some nice scaling properties (Section 4.2.3).

## 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 Appendix B, 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

[2]We stress that probes are ordinary data packets that are part of the data stream and are not explicitly marked. Marking/modifying ordinary packets is undesirable for several reasons: (a) it must be undone by the receiver prior to processing or forwarding, (b) it may cause backward-compatibility problems by introducing packet formats that is are unrecognized by devices along the path, (c) it may run into MTU limitations, etc.

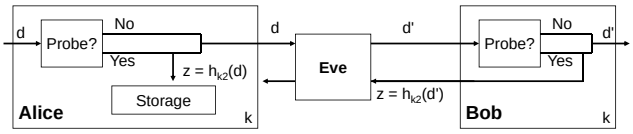


**Figure 1: Secure Sampling.**

using a (packet-hashing) PRF $h$ keyed with $k_1$ and outputting an integer in $\{0, \ldots, 2^n - 1\}$, as follows:

$$\mathsf{Probe}_{k_1}(d) = \begin{cases} \text{YES}, & \text{if } \frac{h_{k_1}(d)}{2^n} < p; \\ \text{NO}, & \text{else.} \end{cases} \quad (3)$$

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 4.1. *The symmetric secure sampling protocol is an $(\alpha, \beta, \delta)$-secure PQM protocol for $\alpha < \beta \leq 4\alpha$ as per Definition 2.1, whenever the probe frequency $p$ and number of packets per interval $T$ satisfy*

$$pT > \ln(\tfrac{1}{\delta}) \tfrac{3}{(\sqrt{\beta}-\sqrt{\alpha})^2} \quad . \quad (4)$$

When $\alpha = \beta/2$ we can use [4, Thm. 19] to 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 PROOF OF THEOREM 4.1. 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 $\mathsf{Probe}$ 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 $\mathsf{Probe}$ 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 2.1, suppose the failure rate is less than $\alpha$. Notice also the false positives condition of Definition 2.1 is conditioned on the fact that no node behaves adversarially, *i.e.,* maliciously drops ACKs (or synchronization messages, see Section B). Thus, 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

$$\begin{aligned} P_{FN} &= \Pr[\, V > pT\sqrt{\alpha\beta} \mid \text{failure rate } < \alpha \,] \\ &= \Pr[\, V > p\alpha T(1 + \tfrac{\sqrt{\beta}-\sqrt{\alpha}}{\sqrt{\alpha}}) |\, E[V] = p\alpha T \,] \\ &\leq e^{-\frac{(\sqrt{\beta}-\sqrt{\alpha})^2}{3}\, pT} \end{aligned} \quad (5)$$

where the equality follows from simple algebra and the fact that when the drop rate is $\alpha$, $V$ is a binomial random variable $\mathcal{B}(\alpha T, p)$, and the inequality follows from the Chernoff bound[5]of [6, Fact 4], which holds when $0 < \frac{\sqrt{\beta}-\sqrt{\alpha}}{\sqrt{\alpha}} < 1$ or $\alpha < \beta < 4\alpha$. 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 2.1. 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. Using [6, Fact 4]again, if probes are sampled independently with probability $p$ then

$$\begin{aligned} P_{FP} &= \Pr[\, V > pT\sqrt{\alpha\beta} \mid \text{failure rate } > \beta \,] \\ &= \Pr[\, V > p\beta T(1 - \tfrac{\sqrt{\beta}-\sqrt{\alpha}}{\sqrt{\beta}}) |\, E[V] = p\beta T \,] \\ &\leq e^{-\frac{(\sqrt{\beta}-\sqrt{\alpha})^2}{2}\, pT} \end{aligned} \quad (8)$$

where the equality follows from simple algebra and the fact that when the drop rate is $\beta$, $V$ is a binomial random variable $\mathcal{B}(\beta T, p)$, and the inequality again follows from the Chernoff bound of [6, Fact 4], which holds when $0 < \frac{\sqrt{\beta}-\sqrt{\alpha}}{\sqrt{\beta}} < 1$ or when $\alpha < \beta$. As observed above, (8) still, holds up to a negligible factor, when the probes are sampled using a PRF. Notice that dropping ACKs (or synchronization messages, see Section B) cannot help Eve, as it only makes the source *more* likely to raise an alarm. It follows from equations (5), (8) and Definition 2.1 that, given $\alpha, \beta$ and $\delta$, such that $\beta < 4\alpha$, the protocol is secure whenever (4) holds. □

[3]When $h$ uses a modified Wegman-Carter construction (see Appendix A),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 4.1) $V$, or the number of unacknowledged probes in Alice's table, is a binomial random variable. We obtain the threshold above using maximum likelihood estimation as $(\mu_\alpha\sigma_\beta + \mu_\beta\sigma_\alpha)/(\sigma_\alpha + \sigma_\beta)$ where $\mu_\alpha = p\alpha T$ is the mean of $V$ when the loss rate is $\alpha T$ and $\sigma_\alpha^2 = (1-p)p\alpha T$ is the variance of $V$ when the loss rate is $\alpha T$, and $\sigma_\beta$ and $\mu_\beta$ are defined analogously. Then, we get the threshold $pT\frac{\beta\sqrt{\alpha}+\alpha\sqrt{\beta}}{\sqrt{\alpha}+\sqrt{\beta}} = pT\sqrt{\alpha\beta}$.

[5]We use the following Chernoff bounds. Let $X_i$ be i.i.d indicator variables with mean $\mu$, and let

$$\Pr\left[\sum_{i=1}^{n} X_i \leq (1-\gamma)N\mu\right] \leq e^{-\gamma^2 N\mu/C_1} \quad (6)$$

$$\Pr\left[\sum_{i=1}^{n} X_i \geq (1+\gamma)N\mu\right] \leq e^{-\gamma^2 N\mu/C_2} \quad (7)$$

If $0 < \gamma < 1$ then [6, Fact 4] gives $C_1 = 2$ and $C_2 = 3$. If $0 < \gamma < \frac{1}{2}$ then [4, Thm. 19] gives $C_1 = C_2 = 2\ln 2$.

## 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 [40] [9,11,14] 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 $\tau$ seconds, where $\tau$ is a constant known to all parties. In Appendix B.2 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 $H(d)$ (computed using a collision-resistant hash) in her table. Suppose Alice sends $T$ packets in the interval. (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 $\tau$ 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 $\mathrm{Sign}_{SK}(u, \mathsf{s}_1(u), \mathsf{s}_2(u))$(see Figure 4.2.1).

4. *(Security check.)* If Alice fails to receive a SaltRelease containing a signature $\sigma$ 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 $\mathsf{Probe}_{\mathsf{s}_1(u)}(z) =$ 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 this step of the protocol can be computed *offline*.

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 4.2. *The RSSS protocol is an $(\alpha, \beta, \delta)$-secure PQM protocol for $\alpha < \beta \leq 4\alpha$ as per Definition 2.1, whenever the probe frequency $p$ and number of packets per interval $T$ satisfy*

$$pT > \ln(\tfrac{1}{\delta}) \tfrac{3}{(\sqrt{\beta}-\sqrt{\alpha})^2} \quad . \tag{9}$$

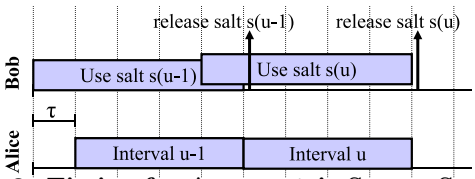


**Figure 2: Timing for Asymmetric Secure Sampling.**

| packet digest | Ack | Probe |
|---|---|---|
| $z_1 = H(d_1)$ | | Yes |
| $z_2 = H(d_2)$ | $\mathrm{MAC}_{\mathsf{s}_2(u)}(B, z_2, u, )$ | 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 |
| $z_8 = H(d_8)$ | $\mathrm{MAC}_{\mathsf{s}_2(u)}(B, z_8, u)$ | Yes |
| $z_9 = H(d_9)$ | | No |
| $z_{10} = H(d_{10})$ | | No |
| $z_{11} = H(d_{11})$ | $\mathrm{MAC}_{\mathsf{s}_2(u)}(B, z_{11}, u)$ | Yes |
| $z_{12} = H(d_{12})$ | | Yes |
| $z_{13} = H(d_{13})$ | | No |

**Figure 3: Alice's table after at the end of interval $u$. Here Alice observes packet-delivery failures for packets $1, 12$.**

When $\beta = 2\alpha$ we can use a tighter bound of [4, Thm. 19](instead of (9)) 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 4.1 (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 $\tau$ 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 $\tau$ 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 [33], secure clock synchronization is a non-trivial problem. In Appendix B.2 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 $T_{\mathrm{B}}$ 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 4.1, the protocol is secure if $\beta < 4\alpha$ and

$$pT_A(B) > \ln(\tfrac{1}{\delta})\tfrac{3}{(\sqrt{\beta}-\sqrt{\alpha})^2} \quad . \tag{10}$$

#### 4.2.3 A note on the scalability of RSSS

Our Receiving-Server Secure Sampling protocol has useful scaling properties, that make it attractive even outside the client-server setting. Consider a network with $M$ routers, where each router would like to run a pairwise PQM protocol (acting as both a sender and receiver) with all other routers. Then, while the Receiving-Server Secure Sampling protocol requires that each router to store the $M-1$ public-keys, the storage and *online* computation overhead at each router is independent of the $M$! To see why, first notice from Section 4.2.1 that when the router acts as a receiver, its incurs no storage overhead, while its computation/communication overhead depends only on the total volume of traffic it receives, and not on the number of senders $M-1$. Next, when the router acts as a sender, it only needs to keep a table of digests of the packets it sends (see *e.g.,* Figure 3) that additionally specifies the receiver the each packet stored in table was sent to. The size of this table depends on the volume of sent traffic, but is independent of $M$. Furthermore, it is computed (using a collision resistant hash) independently of the keys of any of the $M-1$ receivers. Indeed, receiver-specific computations are *only* required during the *offline* 'security check' step of the protocol.

### 4.3 Some sample parameters

Suppose $\alpha = \frac{1}{2}\beta$ and $\beta = 1\%$. We 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 from [4, Thm. 19]on Theorem 4.1 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 (4) 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 (9) 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 (9), 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

(10), 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, usually provides approximate information about the aggregate stream of packets, while a sample provides exact information about a single packet in the stream.) At the end of the interval, Bob sends his sketch to Alice and she compares the sketches to decide whether the failure rate exceeded $\beta$.

We can apply several sketching techniques [3, 5, 13, 48] for *second moment estimation* (or $\ell_2$-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.* [13, 48] and the references in [50]), 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 [13]'s scheme that may be of independent interest.

In this section we start by explaining the relationship between moment 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 second-moment 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 moment estimation

We now show how why $p^{th}$-moment estimation (for $p \geq 1$) is sufficient to realize PQM. Later on, we will use this argument to how show second-moment estimation (for which a number of highly efficient and simple schemes are known [3, 5, 13, 48]) can be used to realize PQM.

**Preliminaries.** Recall that Alice sends a stream of $T$ packets to Bob during an interval and let $U$ be the "universe" of all possible packets (*e.g.,* if packets are 1500 bytes long then $|U| = 2^{1500 \cdot 8}$). We define the *characteristic vector* of a stream to be a $U$-dimensional vector that has $c$ in the position corresponding to packet $x$ if packet $x$ appears in the stream $c$ times (*e.g.,* for the stream 1,2,4,2,2 of packets drawn from universe $U = [4]$ the characteristic vector is [1 3 0 4].) In our setting, a characteristic vector is too long ($2^{1500 \cdot 8}$) to be represented explicitly; we will use it only for the purpose of explaining our protocols. Also, recall that $p^{th}$-moment of a vector $\mathbf{v}$ is $\|\mathbf{v}\|_p^p = \sum_i (v_i)^p$. (Note that the $\ell_p$-norm is just the $p^{th}$ root of $p^{th}$ moment of the vector.)

**Relationship between PQM and the $p^{th}$ moment for $p > 1$.** Let $\mathbf{v}_\mathrm{A}$ be the characteristic vector for the stream of packets sent by Alice, and let $\mathbf{v}_\mathrm{B}$ be the characteristic vector for the stream of packets received by Bob. Now consider the characteristic vector $\mathbf{x} = \mathbf{v}_B - \mathbf{v}_A$. We can decompose any $\mathbf{x}$ into two vectors $\mathbf{x} = \mathbf{d} + \mathbf{a}$. The vector $\mathbf{d}$ is the characteristic vector of packets *dropped* on the path from Alice to Bob, and contains the non-negative components of $\mathbf{x}$. The vector $\mathbf{a}$ is the characteristic vector of packets *added* on the path from Alice to Bob, and contains the non-positive components of $\mathbf{x}$. Also notice that the non-zero coordinates of $\mathbf{d}$ and $\mathbf{a}$ are disjoint. Let $D$ be a count of the number of dropped packets, and $A$ be a count of the number of added packets. We now have the following very useful identity

$$\|\mathbf{x}\|_p^p = \|\mathbf{d}\|_p^p + \|\mathbf{a}\|_p^p = D + \|\mathbf{a}\|_p^p \geq D + A \qquad (11)$$

The first equality follows because the non-zero coordinates of $\mathbf{d}$ and $\mathbf{a}$ are disjoint. The second equality follows because every packet that Alice send is unique so that that $\mathbf{d}$ is a $\{0, 1\}$-vector for every $i \in [K+1]$. Finally, the last inequality follows because $\mathbf{a}$ is an integer vector, so that for any $p \geq 1$, it follows that $\|\mathbf{a}\|_p^p \geq |\mathbf{a}|_1 = A$ with equality when $p = 1$.

Now, Recall Definition 2.1. To satisfy the "few false positives" condition we need to consider the *benign case* in which at most $D \leq \alpha T$ packets are dropped during the interval, and no packets are added so that $\|\mathbf{a}\|_p^p = 0$. From (11) it follows that satisfying the "few false positives" condition (in the benign case), just requires that Alice should not raise an alarm if $\|\mathbf{x}\|_p^p = D + 0 \leq \alpha T$.

To satisfy the "few false negatives" condition we need to consider the *malicious case* in which Eve drops $D \geq \beta T$ packets, and adds an arbitrary number of (potentially non-unique) packets $A \geq 0$. (We think of a packet modification as a dropped packet *plus* an added packet.) From (11) it follows that satisfying the "few false positives" condition (in the malicious case), requires that Alice raise an alarm if $\|\mathbf{x}\|_p^p \geq \beta T$.

The discussion above suggests the following ridiculous PQM protocol: Have Bob and Alice maintain $\mathbf{v}_\mathrm{A}$ and $\mathbf{v}_\mathrm{B}$, and have Bob send Alice $\mathbf{v}_\mathrm{B}$ at the end of the interval. Then define a decision threshold $\Gamma \in [\alpha T, \beta T]$, and have Alice raise an alarm if the first moment $\|\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}\|_p^p > \Gamma$. Notice that, in the malicious case, adding packets doesn't help Eve; whenever Eve adds packets, she simply increases $\|\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}\|_p$, and makes Alice more likely to raise an alarm.

**Sketches.** Of course, the PQM protocols described above are completely impractical; the $\mathbf{v}_\mathrm{A}, \mathbf{v}_\mathrm{B}$ are too large to be stored or transmitted explicitly. This is exactly where *sketching* comes in. A $p^{th}$-moment estimation sketch is a set of probabilistic algorithms that allows us to estimate the $p^{th}$-moment of a vector $\mathbf{v}$ of length $|U|$ from a shorter sketch $\mathbf{w}$ of length $N$; typically, the length of the sketch is depends only on the number of packets in the stream, and not on the size of the universe $U$.

We will concern ourselves with moment estimation schemes where the sketch may be derived from $\mathbf{v}$ via a random linear map, *i.e.,* $\mathbf{w} = R\mathbf{v}$ where $R$ is a *random* $N \times |U|$ matrix drawn from some distribution $\mathcal{S}$. Then an estimator $V$ for $p^{th}$ moment of $v$ is computed from $\mathbf{w}$; in the all schemes we consider here, the estimator will simply be $\|\mathbf{w}\|_p^p$.

Of course, since $R$ is also as long as $|U|$, in our setting $R$ is too large to store explicitly. However, notice that 'sketching' a packet $x$ is exactly equivalent to adding the $x^{th}$ column of $R$ to the sketch $\mathbf{w}$. This suggests the following efficient approach to sketching: initialize $\mathbf{w} = 0$, and for every packet $x$ in the stream, generate the $x^{th}$ column of $R$ by hashing

the packet with $h$ and adding the hash value $h(x)$ to the sketch $\mathbf{w}$. As long $h$ can generate length $N$-vectors that are distributed identically to the column vectors of matrices drawn from $\mathcal{S}$, these is exactly equivalent to computing the sketch via an explicit random linear map $\mathbf{w} = R\mathbf{v}$ .

Thus, we now have a practical PQM protocol based on $p^{th}$-moment estimation for $p > 1$: Alice and Bob share a hash function $h$ and compute $\mathbf{w}_\mathrm{A} = R\mathbf{v}_\mathrm{A}$ and $\mathbf{w}_\mathrm{B} = R\mathbf{v}_\mathrm{B}$ on their streams using the hashing approach described above. Bob then *securely transmits* $\mathbf{w}_\mathrm{B}$ to Alice. Since $\|\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B}\|_p = \|R(\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B})\|_p$; thus, if the sketches *accurately estimate* the first moment, it suffices to raise an alarm if $\|\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B}\|_p^p > \Gamma$ for $\Gamma \in [\alpha T, \beta T]$.

**Dealing with adversaries.** However, our work is not complete. Recall that we would like our PQM protocol to operate correctly in the presence of adversaries on the path. Thus, we still need to discuss what we mean by the terms *secure transmission* and *accurate estimation* in protocol we described above.

Recall that Eve occupies the path between Alice and Bob, and consider the practical PQM protocol that we described above. In the malicious case, there are a number of ways that Eve could attempt to bias the results of this protocol.

- Eve could try to convince Alice than $\beta T$ packet drops occurred by altering the sketch $\mathbf{w}_\mathrm{B}$ that Bob sends to Alice. Preventing this attack is simple; we shall require that Bob send his sketch $\mathbf{w}_\mathrm{B}$ to Alice in a message that is authenticated with a MAC.

- Next, observe that because Eve occupies the path between Alice and Bob, she has the power to choose which packets Bob receives in his stream. Thus, if the adversary can predict the outputs of the hash function $h$ used to map packets to the sketch, she can choose to add and drop packets to Bob's stream in a way that cannot be detected by Alice! For instance, Eve could drop some set of packets and replace them with a different set of packets that map to the sketch in an identical way.

  Typically, the correctness of $p^{th}$-moment estimation schemes relies on the fact that the randomness used for sketching (*i.e.,* to choose the hash function $h$) is chosen *independently* of the stream to be sketched. However, in our setting, this is not necessarily the case; if the hash function $h$ is public, then adversary choice of Bob's stream $\mathbf{v}_\mathrm{B}$ can *depend* on the randomness used for sketching, $h$. (This observation was independently made by Mironov, Naor, and Segev [34].) For this reason, we replace the public hash function $h$ used for sketching with a *keyed* hash function $h_{k_u}$, keyed with a secret key $k_u$ is shared between Alice and Bob, and is refreshed every interval.

## 5.2 The secure sketch protocol

We are finally ready to describe our secure sketch PQM protocol. Our protocol works in intervals. We assume Alice and Bob share a secret (master) key $(k_1, k_2)$, and derive an interval key $k_u$ for each interval $u$ (see Section 3). In Appendix B, we provide a detailed treatment of techniques that Alice and Bob can use to synchronize their intervals. Within interval $u$, our secure sketch protocol operates as follows:

1. *(Sketch.)* Alice runs a sketching algorithm, using a keyed hash $h_{k_u}(\cdot)$ keyed with secret interval key $k_u$ to incrementally compute a sketch $\mathbf{w}_\mathrm{A}$ of the vector $\mathbf{v}_\mathrm{A}$ induced by the packet it sends. Bob similarly uses $h_{k_u}(\cdot)$ to compute sketch $\mathbf{w}_\mathrm{B}$ of the vector $\mathbf{v}_\mathrm{B}$ induced by the packets he receives.

2. *(Interval End.)* After sending the $T^{th}$ packet in the interval, Alice sends an 'Interval End' message to Bob, authenticated with the master key $k_1$, and containing her sketch $\mathbf{w}_\mathrm{A}$ and the next interval number $u + 1$. She then refreshes her sketch (*i.e.,* sets $\mathbf{w}_\mathrm{B} = 0$ ) and refreshes the interval key (*i.e.,* computes $k_{u+1}$ using a PRF keyed with the master key $k_2$ as described in Section 3).

3. *(Report.)* Upon receiving the 'Interval End' message and verifying the correctness of its MAC, Bob computes the 'difference sketch' $\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B}$, and sends a 'Report' message to Alice, authenticated with the master key $k_1$, containing the 'difference sketch' $\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B}$, and the current interval number $u$. Bob then refreshes his sketch and computes the interval key for the next interval $u + 1$.

4. *(Security Check.)* Upon verifying the MAC on the 'Report Message', Alice uses the difference sketch $\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B}$ to compute an estimate $V$ of $\|\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}\|_2^2$ and raises an alarm if and only if $V > \Gamma = 2\alpha\beta T/(\beta + \alpha)$.

Our protocol has a number of attractive properties. First, notice the we require the transmission of only two control messages ('Interval End', and 'Report'), and no packet modifications. Second, notice that the 'Security Check' phase can be computed offline. Finally, notice that Alice and Bob need only store single sketch at any given time; at the end of each interval, Alice and Bob immediate transmit their sketches as control messages, refresh their sketches, and begin monitoring a new interval.

## 5.3 Security of the secure sketch protocol

We formalize the intuition of Section 5.1 with the following theorem.

THEOREM 5.1. *Suppose that the sketch algorithm guarantees, that if the hash function used for sketching is chosen randomly and independently of* $\mathbf{v}$*, then with probability at least* $1-\delta$*, the estimate of the* $p^{th}$*-moment* $\|\mathbf{v}\|_p^p$ *for* $p \geq 1$ *is within* $(1 \pm \varepsilon)$ *for* $\epsilon = \frac{\beta-\alpha}{\beta+\alpha}$*. Then, the secure sketch protocol is a* $(\alpha, \beta, \delta)$*-secure PQM protocol as per Definition 2.1.*

PROOF PROOF OF THEOREM 5.1. Consider the *malicious* case. First observe that Eve cannot forge the 'Interval End' or 'Report' control messages, since the control messages are authenticated using a secure MAC (and dropping the report will only cause Alice to raise an alarm). It follows that, for both the *benign* and *malicious* case, Alice gets a consistent version of the difference sketch $\mathbf{w}_\mathrm{A} - \mathbf{w}_\mathrm{B}$ at the end of the every interval.

Now, observe that (a) no effect of the hash function $h_{k_u}$ is visible to Eve until after the interval ends, and (b) $k_u$ is kept secret from Eve and thus chosen (pseudo)randomly and independently of $\mathbf{v}_\mathrm{A}$ and $\mathbf{v}_\mathrm{B}$. It follows that the sketching algorithm generates a $(1 \pm \varepsilon)$-estimate of $V$ of $\|\mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}\|_2$ with probability $1 - \delta$. Thus, letting $\mathbf{x} = \mathbf{v}_\mathrm{A} - \mathbf{v}_\mathrm{B}$, we have:

1. No false positives: if $D \leq \alpha T$ and $A = 0$, then as discussed in Section 5.1 it follows that $\|\mathbf{x}\|_p^p = \|\mathbf{d}\|_p^p = D \leq \alpha T$. Now, with probability $1-\delta$ we have that the estimate

$$\begin{aligned} V &\leq (1+\varepsilon)\|\mathbf{x}\|_p^p \\ &\leq 1 + \tfrac{\beta-\alpha}{\beta+\alpha})\alpha T \\ &= \tfrac{2\beta\alpha}{\beta+\alpha} T = \Gamma \end{aligned}$$

2. No false negatives: if $D > \beta T$, then as discussed in Section 5.1 it follows that $\|\mathbf{x}\|_p^p = \|\mathbf{d}\|_p^p + \|\mathbf{a}\|_p^p > \|\mathbf{d}\|_p^p > \beta T$. Similarly, with probability $1-\delta$, it follows that 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 probability $1-\delta$ Alice can use the decision threshold $\Gamma$ to decide between cases where $D < \alpha T$ and $D > \beta T$. $\square$

Recall that $\mathbf{d} \in \{0,1\}^U$ because Alice sends unique packets. A closer look at the proof shows it suffices if the sketch guarantees that (a) the estimate is at most $(1+\varepsilon)\alpha T$ for vectors that have all entries in $\{0,1\}$ and with norm $\|\mathbf{v}\|_p^p \leq \alpha T$, and (b) the estimate is at least $(1-\varepsilon)r$ for vectors $\mathbf{v}$ that have at least $r \geq \beta T$ 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 5.2 below.

**Turning the protocol on and off.** In order to reduce resource consumption, it sometimes makes sense for a router to 'turn off' the secure sketching protocol. However, an adversary could take advantage of the fact the protocol is 'off' for certain intervals in order to bias monitoring results, selectively dropping packets when the protocol is 'off', and behaving itself while the protocol is 'on'. Thus, it is crucial to ensure that intervals when the protocol is 'on' indistinguishable from intervals when the protocol is 'off'.

Notice that from Eve's perspective, the only indication that the protocol is 'on' are the two control messages ('Interval End' and 'Report'). Thus, while in an 'off' interval, Alice and Bob need not compute hashes over packet contents or to maintain sketches (so that there are significant savings in storage and computation), we still require the appropriate control messages to be sent. In an 'off' interval, we require (a) Alice to count the number of packets she sends to Bob and send a dummy 'Interval End' message each time the counter reaches $T$, and (b) Bob to respond with a dummy 'Report' packet. To make the dummy control messages indistinguishable from real control messages, we will also require (c) that *all* control messages sent by the protocol are encrypted and padded to a fixed length.

With this approach, a sender with the resources to run only $K$ instances of the 'secure sketch' protocol, can engage in PQM with $M > K$ receivers by choosing a random set of $K$ of $M$ receivers for which the protocol should be 'on' in a given interval. Note that that selection of 'on' intervals should be *random*, in order to prevent an adversary from selectively attacking the 'off' intervals by using side-channel information (*e.g.,* observing if the sender switches to a new path) to distinguish between which intervals that 'on' or 'off'.

## 5.4 Plugging in sketching schemes

In this section we show how to instantiate our PQM protocol with known $p^{th}$-moment estimation sketching schemes, such that the schemes satisfy the requirements of Theorem 5.1. We will focus on two highly efficient schemes for estimating the *second*-moment: the 'classic' sketching technique [3,5] based on the Johnson-Lindenstrauss lemma, and the more efficient 'CCF' sketch of Charikar, Chen and Farach-Colton [13].

In each scheme, packet hashing can be done with either 4-wise independent hash function or PRF. We consider both cases. While 4-wise independent hash functions are theoretically faster than PRFs (see also the discussion in Section 3),using these weaker hash function comes at the cost of worse sketch parameters $N$. Both schemes operate by taking a single pass over the data stream to compute the sketch $\mathbf{w}$, and compute the estimator as $V = \|\mathbf{w}\|_2^2$. When packet-hashing is done with *either* a 4-wise independent hash function *or* a PRF [3, 5, 48], both schemes have estimators $V$ with expectation $\|\mathbf{v}\|_2^2$ and variance $\frac{2}{N-1}\left(\|\mathbf{v}\|_2^4 - \|\mathbf{v}\|_4^4\right)$.

We next describe each scheme, and show how they 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.

### 5.4.1 Classic Second-Moment Estimation Sketches

Alon, Matias, and Szegedy [5] suggest the following approach to sketching: when receiving a packet $d$ map it to a vector $b \in \{-\frac{1}{N}, \frac{1}{N}\}^N$ and add $b$ to the sketch $\mathbf{w}$. Thus, the update time per incoming packet is exactly $N$, the number of bins in the sketch.

**Packet hashing with a 4-wise independent hash.** Alon et. al [5] require a hash function such that each entry of $b$ in 4-wise independent, and every entry of $b$ is completely independent of all other entries of $b$. They then show how to estimate $\mathbf{v}$ within $(1 \pm \varepsilon)$ with probability $\delta$ using a sketch with $N \geq \frac{2}{\varepsilon^2\delta}$ bins.

**Packet hashing with a PRF.** Achlioptas [3] obtained a bound on $N$ by requiring each entry of $b$ to be computed using an (independent) PRF producing either $+\frac{1}{N}$ or $-\frac{1}{N}$ with probability $\frac{1}{2}$. Achlioptas showed that obtaining an $(\varepsilon, \delta)$-approximation of the second moment requires

$$N > \tfrac{12}{\varepsilon^2} \tfrac{1}{3-2\varepsilon} \ln \tfrac{1}{\delta} \tag{12}$$

bins in the sketch. Notice the that PRF approach requires $O(\log \frac{1}{\delta}/\delta)$ less storage then the 4-wise independent hashing approach.

**Sizing each bin in the sketch.** To prevent overflow, we can take each bin in the sketch to hold integers in $[-K, +K]$ where $K = \sqrt{2T \ln(\frac{200N}{\delta})}$, so that each bin requires $1 + \log_2 K$ bits of storage. [6](We can also change the protocol to raise an alarm if any bin overflows, since this will happen with low probability in the benign case.)

[6]When we store the sketch, we drop the $\frac{1}{N}$ factor, and find $K$ such that the probability that each bin overflows is at most $\frac{\delta}{N}\frac{1}{100}$. If $X_i$ is an indicator variable that equals 1 with probability $\frac{1}{2}$ and $-1$ otherwise, then the count in each bin is the random variable $X = \sum_{i=1}^{T} X_i$. Then, from the Chernoff bound we have that $\Pr[\,|X| \geq K\,] \leq 2\exp(-\frac{K^2}{2T}) \leq \frac{\delta}{100N}$. Finally, we get $K = \sqrt{2T \ln(\frac{200N}{\delta})}$.

### 5.4.2 CCF second-moment estimation sketch.

The sketch of Charikar, Chen, and Farach-Colton [13] can be adapted [48] to give a second-moment estimation algorithm with a faster update time; instead of updating *all* $N$ bins each time a new packet arrives as in the classic sketch, the CCF scheme only updates a *single* bin. In our context, the CCF update algorithm requires that each incoming packet $d$ is hashed to a pair $(i, b)$ where $i \in [N]$ and $b \in \{\pm 1\}$, and $b$ is added to the $i^{th}$ bin in the sketch $\mathbf{w}$. To prevent overflow in each bin, we take each bin to hold integers in $[-K, +K]$ where $K = 2\sqrt{\frac{T}{N}\ln(\frac{200N}{\delta})}$. Thus, we require [7]

$$1 + \tfrac{1}{2}\log_2\left(4\tfrac{T}{N}\ln(\tfrac{200N}{\delta})\right) \quad \text{bits / bin} \tag{13}$$

**Packet-hashing with 4-wise independent hashes.** In Appendix E.1 we show that if the incoming packet $d$ is hashed with two independent 4-wise independent hash functions, ( $i$ is computed using a 4-wise independent hash with output domain $[N]$ and $b$ is computed using a 4-wise independent hash with output domain $\{-1, 1\}$), then we require at $N \geq \frac{2}{\varepsilon^2\delta}$ bins in our sketch.

**Packet-hashing with a PRF.** In the general case, the faster update time of CCF comes at the cost of increased storage. More precisely, in order to get a $(1 \pm \epsilon)$ accuracy with probability $1 - \delta$, the CCF schemes require a larger $N = \Theta(\frac{1}{\delta\epsilon^2})$, rather than $N = \Theta\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ of the classic scheme. In the general case this is true even is hashing is performed with a truly random function![8] We show why with the following counterexample: consider the vector $\mathbf{v} = 10^{10}e_x + 10^{10}e_{x'}$ where $e_x$ is the vector with 1 in coordinate $x$ and zero elsewhere, and $x \neq x'$. Then $\|\mathbf{v}\|_2^2 = 2 \cdot 10^{20}$, but with probability $1/2N$ a sketch of $\mathbf{v}$ will be 0.

Fortunately, it turns out that, in our setting, the CCF scheme need not incur the cost of increased storage! We now prove that in our setting the CCF scheme we can have $N = O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)$. This follow because, per the discussion after Theorem 5.1, (a) we assume that Alice sends unique packets, and (b) we only care about deciding whether $\|\mathbf{v}\|_2^2$ lies above or below a threshold, rather than getting an accurate estimate of $\|\mathbf{v}\|_2^2$.

Our theorem requires supposes that packet hashing is performed using an two independent random function: one to chose $i \in [N]$ and another to choose $b \in \{-1, 1\}$. When the CCF algorithm uses a random function for hashing, we can think of the sketch $\mathbf{w}$ as computed from the characteristic vector $\mathbf{v}$ via a random linear map, *i.e.,* $\mathbf{w} = R\mathbf{v}$, where $R$ is chosen *uniformly at random* from set $\mathcal{S}_{\text{CCF}}$. For the CCF algorithm, $\mathcal{S}_{\text{CCF}}$ is the set of $N \times |U|$ matrices where each column has $\pm 1$ in some row and zeros everywhere else. Hence, we have the following theorem:

THEOREM 5.2. *For any vector $\mathbf{v} \in \mathbb{Z}^U$, choose the $N \times U$ matrix $S$ uniformly from $\mathcal{S}_{\text{CCF}}$ and set $\mathbf{w} = S\mathbf{v}$. Then, for all $\varepsilon \in [0, 1)$ and $\eta$ such that $\left(\frac{1-\eta}{1+\eta}\right)^2 = \max\left(\frac{1+\frac{\varepsilon}{2}}{1+\varepsilon}\ ,\ \frac{1-\frac{3\varepsilon}{4}}{1-\frac{\varepsilon}{2}}\right)$, choosing*

$$N \geq \tfrac{24}{\varepsilon^2}\ln\tfrac{2}{\delta} \tag{14}$$

$$q, r \geq \tfrac{3N}{\eta^2}\ln\tfrac{4N}{\delta} \tag{15}$$

*ensures that the following two items occur with probability at least $1 - \delta$:*

1. *If $\mathbf{v} \in \{-1, 0, 1\}^U$, and $\|\mathbf{v}\|_2^2 \leq q$, then $\|\mathbf{w}\|_2^2 < (1 + \varepsilon)q$.*
2. *The number of* non-zero *entries in $\mathbf{v}$ is $r$, then $\|\mathbf{w}\|_2^2 > (1 - \varepsilon)r$.*

See Appendix E.2 for Theorem E.4 a tighter and more precise statement of Theorem 5.2, as well as its proof. Notice the theorem requires conditions on both the number of non-zero elements in $\mathbf{v}$ and $N$. The fact that $N$, the number of bins in the sketch, must be large is not so surprising. We need $\mathbf{v}$ to have many non-zero elements 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.

**Applying Theorem 5.2.** To apply the theorem into our setting, assume that the PRF used for packet hashing is indistinguishable from a random function. Then, set $\varepsilon = \frac{\beta-\alpha}{\beta+\alpha}$, set $\mathbf{x} = \mathbf{v}_{\text{A}} - \mathbf{v}_{\text{B}}$, and set $q = \alpha T$. The false positive condition is satisfied because we have $\mathbf{x} \in \{0, 1\}^U$ and $\|\mathbf{x}\|_2^2 = |\mathbf{x}|_1 = D \leq \alpha T$, so with probability $1 - \delta$,

$$V = \|\mathbf{w}\|_2^2 < (1 + \varepsilon)\|\mathbf{x}\|_2^2 \leq (1 + \varepsilon)\alpha T = \tfrac{2\alpha\beta}{\alpha+\beta}T$$

The false negative condition is satisfied because we have the number of drops is $r > \beta T$. So, with probability $1 - \delta$, we get that

$$V = \|\mathbf{w}\|_2^2 > (1 - \varepsilon)r \geq (1 - \varepsilon)\beta T = \tfrac{2\alpha\beta}{\alpha+\beta}T$$

where the first inequality comes from the fact that $\mathbf{x} = \|\mathbf{d}\|_p^p + \|\mathbf{a}\|_p^p$ and $\|\mathbf{d}\|_p^p$ is a $\{0, 1\}$-vector with $r$ entries that are $+1$.

### 5.4.3 Some sample parameters and experiments

In the following, we use the following sample parameters: We suppose the detection threshold is $\beta = 0.01$, the false alarm threshold is $\alpha = \beta/2$ and about $T = 10^7$ packets are sent during an interval. We will require a confidence of $1 - \delta = 99\%$.

[7] We find $K$ such that the probability that each bin overflows is at most $\frac{\delta}{N}\frac{1}{100}$. If $X_i$ is a random variable that equals 1 with probability $\frac{1}{2N}$, $-1$ with probability $\frac{1}{2N}$, and 0 otherwise, then the count in each bin is the random variable $X = \sum_{i=1}^{T} X_i$. Then, adapting the Chernoff bound that appears in Levchenko [30], we have that $\Pr[\,|X| \geq K\,] \leq 2\exp(-\frac{K^2}{4T\text{VAR}[X_i]}) \leq \frac{\delta}{100N}$. Finally, we get $K = 2\sqrt{\frac{T}{N}\ln(\frac{200N}{\delta})}$ since $\text{VAR}[X_i] = 1/N$.

[8] CCF's [13] sketch can attain better success probability (even with 4-wise independent hashing) 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 $M$.

[9] This analytic bound on $N$ also requires that $\alpha T > \frac{3N}{\eta^2}\ln\frac{4N}{\delta}$ for $\eta$ such that $\left(\frac{1-\eta}{1+\eta}\right)^2 = \max\left(\frac{1+\frac{\varepsilon}{2}}{1+\varepsilon}\ ,\ \frac{1-\frac{3\varepsilon}{4}}{1-\frac{\varepsilon}{2}}\right)$. See Theorem 5.2.

| Scheme | Packet Hashing | $N$, Bins in Sketch | Bits/bin |
|---|---|---|---|
| Classic | 4-wise independent | $\frac{2}{\varepsilon^2\delta}$ | $1+\frac{1}{2}\log_2\left(2T\ln(\frac{200N}{\delta})\right)$ |
| | PRF | $\frac{12}{\varepsilon^2}\frac{1}{3-2\varepsilon}\ln\frac{1}{\delta}$ | |
| CCF | 4-wise independent | $\frac{2}{\varepsilon^2\delta}$ | $1+\frac{1}{2}\log_2\left(4\frac{T}{N}\ln(\frac{200N}{\delta})\right)$ |
| | PRF | $\frac{24}{\varepsilon^2}\ln\frac{2}{\delta}$ [9] | |

**Table 1: (Analytically-derived) parameters for secure sketch PQM.**

| | **Bins in Sketch** | **Sketch Size** | | | | |
|---|---|---|---|---|---|---|
| $\beta/\alpha$ | **N** | $T=10^4$ | $T=10^5$ | $T=10^6$ | $T=10^7$ | $T=10^8$ |
| 2 | 128 | 128B | 144B | 176B | 208B | 208B |
| 4 | 64 | 64B | 88B | 88B | 104B | 104B |
| 8 | 32 | 36B | 48B | 48B | 52B | 52B |
| 16 | 32 | 36B | 48B | 48B | 52B | 52B |
| 32 | 32 | 36B | 48B | 48B | 52B | 52B |
| 64 | 32 | 36B | 48B | 48B | 52B | 52B |

**Table 2: Minimum $N$ bins per sketch, when $N$ is taken as a power of 2, computed via numerical experiments for PQM using CCF with a PRF for packet hashing. Sketch size is computed by multiplying experimentally-obtained value for $N$ with the value obtained from equation (13). We fix $\beta=\delta=1\%$.**

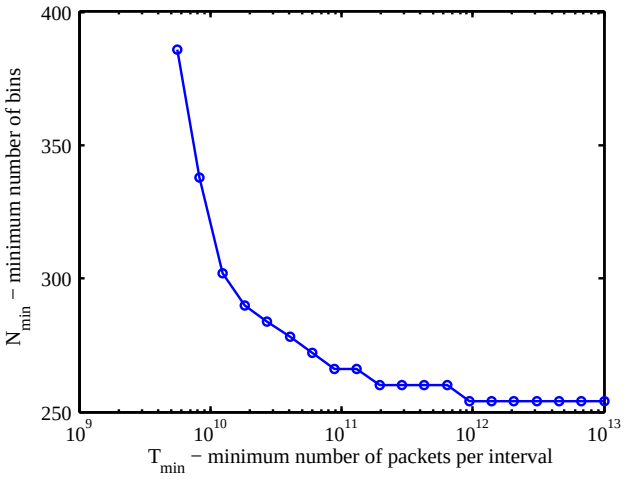


**Figure 4: Theorem E.4 is used to obtain bounds on sketch size, $N$ for a given choice of $T_{min}$, the minimum number of packets per interval. Here $\delta=\beta=2\alpha=1\%$.**

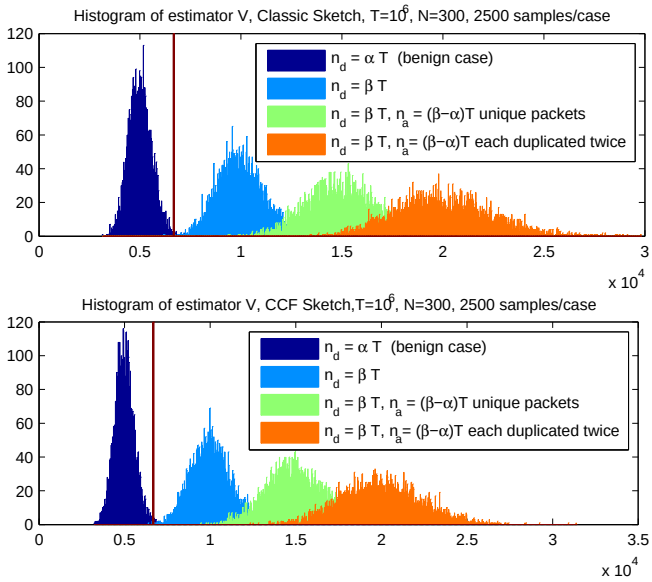


**Figure 5: Histogram of estimator for the (a) classic, and (b) CCF schemes, each using packet-hashing with a PRF and with $N=300$, $T=10^6$, $\beta=2\alpha=1\%$ and threshold $\Gamma=6667$. Histogram computed via numerical experiments.**

**Comparing analytic results.** Combining these sample parameters with the analytic results summarized in Table 3, we see that when 4-wise independent hashing is used, both classic and CCF sketching require $N = 1800$ bins in the sketch. However since CCF requires only 10 bits/bin compared to the 16 bits/bin required for classic sketching, we see that the CCF requires smaller sketches (3.6KB sketch for classic, 2.25KB for CCF sketching). Storage become noticeably smaller when we use a PRF. From Table 3, when a PRF is used we find that the classic scheme requires $N = 214$ bins with 15 bits/bin for a sketch of size 400B. For the CCF scheme, we can apply the refined version of Theorem 5.2 in Appendix E.2 to obtain bounds on $N$, the number of bins in the sketch, for different values of $T_{min}$, the minimum number of packets per interval. We did this in Figure 5.4.2 for $\beta = \delta = 2\alpha = 1\%$, and we found that if there are at least $T_{min} = 1.2 \times 10^{10}$ packets in the interval, we can use $N = 300$ with counters of $b = 14$ bits if we take intervals containing at least $T = 10^9$ packets.

Our Theorem 5.2 for CCF with a PRF introduces an awkward bound on $T_{\min}$, the minimum number of packets that must be sent per interval. However, we believe that this bound is an artifact of our proof technique. As we discuss below, our numerical experiments for CCF with a PRF indicate (though do not conclusively prove) that even $N = 300$ bins suffices even if we use much shorter interval lengths, for instance $T = 10^4$.

**Numerical experiments: histograms.** Figure 5 is a histogram of the classic and CCF estimators $V$ for (from left to right) the benign case where $D = \alpha T$ (here we want the estimator to be below the threshold $\Gamma$ so that Alice does not raise an alarm), and for three cases where $D = \beta T$ 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 $\Gamma$ clearly distinguishes between cases where $D = \beta T$ and the benign cases where $D = \alpha T$. Also, notice if Eve adds packets to the link, she only increases the probability that Alice raises an alarm, as predicted by equation (11). Figure 5 also suggests that taking $N = 300$ suffices for CCF even if we have shorter interval lengths of $T = 10^6$.

**Numerical experiments: CCF with a PRF.** We further studied the CCF with PRF approach by performing a number of numerical experiments to determine $N$, the number of bins in the sketch. In every experiment, we chose $N$ as a power of 2.[10]C code for these experiments is available by request. Our results are presented in Figure 2. Firstly, from Figure 2 we see that $N$ varies with the ratio $\beta/\alpha$, which confirms the analytic results summarize in Figure 5.4.2. However, our numerical experiments suggest that as long as there are $T > 1/\alpha$ packets/interval, the choice of $T$ does not really impact the value of $N$; for a given $\beta/\alpha$ ratio, the minimum choice of $N$ as power of 2 was the same for any value of $T$ ranging from $10^4$ to $10^8$. Next, Figure 2 indicates that sketch size grows with $T$; this growth is logarithmic in $T$ (as expressed by the equation for the number of bits / bin for CCF in equation (13)).

[10]We take $N$ to be a power of two because this makes packet hashing in the CCF sketch more convenient. That is, if $N = 2^\eta$ and we use PRF that produces $\eta$ (pseudo)random bits, then the binary representation of these $\eta$ bits uniformly choose an element of $[N]$. However if $N$ is not a power of 2, a more complicated mapping of these $\eta$ bits is required to uniformly choose an element of $[N]$.

### 5.4.4 Reducing computation with pre-hashing

In our setting, hash function computation can be expensive because we need to compute a PRF (or 4-wise independent) hash over up to $|U| = 2^{1500 \cdot 8}$ bits of the packet. While Appendix A discusses generic techniques for fast per-packet PRF computation based on efficient $\varepsilon_g$-almost universal hash functions, it turns out that we can have even faster hashing constructions for our sketching protocols.

To do this, we again use $\varepsilon_g$-almost universal hash functions. We reduce the cost of PRF computation by first mapping packets from $U$ to a short $n_1$-bit string using the efficient $\varepsilon_g$-almost universal hash function, and then using a PRF (or 4-wise independent) hash to map from $n$-bit to the sketch. Thus, if $n$ is sufficiently small, this means that our PRF can be constructed using a *single* invocation of a block cipher like AES. (Similarly, our 4-wise independent hash need only operate on small number of inputs, making it possible to realize using only *e.g.,* three $n$ bit multiplications.). In Appendix D, we adapt the analysis in [48] to show that an $(\alpha, \beta, \delta)$-secure PQM protocol requires a $\varepsilon_g$-almost universal hash function with

$$\varepsilon_g \leq \frac{\delta}{10^3 T} \frac{\beta - \alpha}{\beta + \alpha} \tag{16}$$

**Sample parameters using GHASH.** Consider using GHASH [32] as our $\varepsilon_g$-almost 2-wise independent hash function. Suppose GHASH produces outputs of length $n$, and each packet is at most 1500B long, and block lengths are $m$, where $m$ is taken as a power of two. Since $\varepsilon_g = \frac{1500 \cdot 8}{m} 2^{-n}$, for $T = 10^7$ packets/interval, and $\delta = \beta = 2\alpha = 1\%$, applying (16) we find that it suffices to choose GHASH with $n = m = 64$ bits. This choice of $n$ is quite short! (For most other applications, GHASH requires block lengths of $m = 128$ bits. Indeed, in Appendix A we found that we needed $m = 128$ bits blocks when we analyze it as part of the PRF.) Thus, it follows that (a) hardware implementations of GHASH will be fast, and (b) PRF computation (to map $n$-bit strings to the sketch) amounts to a single invocation of AES.

### 5.4.5 Other sketches.

**TZ Sketch.** Thorup and Zhang [48] 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 $+1$. (While the update algorithm in TZ is as in the Count-Min sketch [15], the analysis there is different.) However, their second-moment estimation scheme requires a larger bin size (roughly twice the number of bits/bin) than CCF, so we don't consider this scheme any further.

**Other $p^{th}$-moment estimation schemes.** As discussed in Section 5.1, any $p^{th}$-moment estimation protocol, for $p \geq 1$, can be plugged into our protocol to achieve PQM. However, it turns out that the algorithms for second-moment estimation are preferable in our setting because they have the simplest packet-hashing algorithms. For instance, for the classic and CCF sketches, the packet-hashing algorithm amounts to choosing integers in $\{-1, 1\}$ and thus can be efficiently implemented in high speed routers. *c.f.,* with first moment estimation protocols that require choosing real numbers from a Cauchy distribution [27].

**Relationship to the adversarial sketch model.** In concurrent work, Mironov et.al. [34] considering a setting which Alice and Bob are required to sketch *adversarially-chosen* sets, and then compute metrics on their sets after exchanging sketches over a secure channel; their model maps directly to our PQM model, where Alice and Bob's sets (*i.e.,* packet streams) may be chosen adversarially, and then sketches are exchanged via an authenticated channel. Our work deals with the fact that streams are chosen adversarially by requiring Alice and Bob to compute their sketches using shared secret keys. However, Mironov et.al require that sketching is performed *without* any shared randomness. Their approach has significant advantages, including reduced key-management overhead, and extensions to the client-server setting. For instance, a server can 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 as in the asymmetric protocol of Section 4.2. Indeed, following the discussion in Section 5.1, any protocol for $p^{th}$-moment estimation (where $p > 1$) in the adversarial sketch gives an PQM protocol (for the client-server setting) as well. However, the lack of shared randomness in the model of Mironov *et al.*comes at a significant cost; they show that any moment estimation protocol for sets of size $T$ requires at least $\Omega(\sqrt{T})$ storage at Alice and Bob. Thus, these protocols are much less efficient than the $O(\log T)$-storage *keyed* sketches that we considered here.

#### 5.4.6 Summary

For large interval lengths $T$, our analytic results show the most efficient (in terms of storage and update time) instantiation of our secure sketch protocol uses CCF's second-moment estimation scheme [13] with a PRF for packet hashing. Furthermore, our numerical experiments suggest that this is the case even for smaller values of $T$. While in theory, using a PRF instead of 4-wise independent hash is more computationally expensive, in practice, this not usually the case (see discussion in Section 3). Furthermore, we also showed how to reduce the computational cost of computing a PRF on each packet by pre-hashing packets with an $\varepsilon_g$-almost universal hash function, and then applying a single invocation of a fast PRF.

## 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 2.1 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 [47], is insecure according to Definition 2.1.

### 6.1 Keys are necessary

We argue that there must be some form of shared secret information between Alice and Bob. To see that keys are necessary, 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.

We emphasize that this argument only proves that one of the parties (Alice or Bob) has some secret, while the other party holds some information that depends on that secret. For instance, Alice and Bob could share symmetric keys, or Bob might have a public-private key pair $(PK, SK)$ while Alice has the public key $PK$,

We further remark that the necessity of keys holds only if Eve has the power to *add* packets. However, we believe that it is unrealistic to assume that the adversary cannot add even a single packet; indeed, the security of some protocols (*e.g.,* our secure sketch protocol) can be broken if the adversary successfully forges (*i.e.,* adds) a single 'Report' packet!

### 6.2 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 [19] 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 [26]. 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.

A Keyed Identification Scheme (KIS) is a challenge-response protocol in which the two parties share a secret key, and Alice wants to verify Bob's identity. To do this, Alice typically sends Bob a challenge, that Bob must respond to using his secret key. A KIS is secure if Percy, an impersonator who eavesdrops on the interactions between Alice and Bob but does not know the secret key, cannot impersonate Bob by come up with a correct response to the challenge (with probability better than just randomly guessing the response).

We use a *reduction* to prove that *any* PQM scheme that is secure according to Definition 2.1 is at least as complex as KIS. First, we show that given any secure PQM protocol, we can construct a secure KIS. The construction is simple: the challenge in the KIS are the $T$ packets that Alice sends to Bob during an interval of the PQM protocol. The correct response in the KIS is the acks/reports that Bob sends to Alice during an interval of the PQM protocol. Next, we show that if the PQM scheme used in the above construction is secure according to Definition 2.1, then our KIS construction is also secure. We do this in contrapositive, by showing that if there existed an efficient adversary Percy that breaks the security of this KIS construction, then Percy can be used to construct an adversary Eve that breaks the security of the PQM protocol. To do this, we show how Eve can break the security of the PQM protocol if she is given access to Percy: First, whenever Percy wants to eavesdrop an interaction between Alice and Bob, Eve lets Percy observe an interval of the PQM protocol. Next, when Percy is ready to impersonate Bob, Eve gives the $T$ packets that Alice sends to Bob to Percy as his KIS challenge, but now, instead of forwarding Alice's packets on to Bob, Eve *drops*

$T$ packets and instead responds to Alice with Percy's KIS response. The proof follows from the fact that Alice will not raise an alarm (and therefore Eve breaks the security of the PQM protocol) whenever Percy produces a successful response the challenge in the KIS (and therefore breaks the security of the KIS).

**Remark.** One could hope for the stronger statement that some kind of cryptographic operation is necessary for *every packet sent by Alice*. However, this is false. Indeed, consider a secure sketch PQM protocol that uses the first-moment estimation protocol of Mironov *et al.* [34], as discussed in Section 5.4.5. Then, we have a PQM protocol that uses no cryptographic computations for packet hashing, and only uses a two cryptographic operations per interval, *i.e.,* computing the MACs on the 'Interval End' and 'Report' packets sent the end of the interval.

# 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 [7] and IPsec. We argue that obtaining PQM protocols that perform well for one particular objective often involves trading off some other objective.

## 7.1 A broader space of design objectives

*Marking packets.* We prefer protocols that do not modify any packets sent by the source edge-network, *e.g.,* by packet marking or encryption. This approach has the advantage of allowing the PQM protocol to be backwards compatible with IP, not increasing packet size, minimizing latency in the router, and allowing the source to turn the PQM protocol on and off without having to coordinate with the destination. Furthermore, avoiding packet marking also means we can implement the PQM protocol in a monitor located off the critical packet-processing path in the router.

*Estimating delay.* We prefer protocols that allow Alice to estimate round-trip delay, without making assumptions about the clock synchronization between Alice and Bob.

*Feedback latency.* We prefer protocols that perform well for small interval lengths, so that Alice need only send a small number of packets before she has sufficient information to decide whether or not to raise an alarm. In general, due the high variance in network conditions, it is better to avoid making routing decisions using measurement made over short timescales [42]. However, an PQM protocol that provides fast feedback empowers the edge network to react quickly when situations are particularly dire (*i.e.,* when a path fails completely). Furthermore, fast feedback can be used to detect transient faulty conditions, and can be used when enforcing SLAs to ensure that repeated, short periods of poor performance are not detected because the PQM protocol uses large interval lengths.

*Client-server* v.s., *peers.* We consider both (a) the *peer* setting, where the source and destination can devote equivalent computational resources to the protocol, (*e.g.,* a corporation that wants to ensure availability between a pair of sites in geographically-disparate locations), and (b) the *client-server setting*, where one party can devote more resources to the protocol (*e.g.,* a client wanting to ensure that his packets are correctly delivered at a web server).

*Symmetric vs. public keys.* Per our negative results in Section 6, all of our protocols require some sort of cryptographic key infrastructure. However, there are many settings, (*e.g.,* when a client has only a very short connection with a web server), where we prefer to design protocols that do not require a handshake protocol between each source-destination pair in order to generate a symmetric key. Furthermore, when one edge network runs PQM protocols with *multiple* other edge networks, it is extremely useful to have protocols that allow an end-point run concurrent PQM protocols using a *single key* (*e.g.,* a public key). This way, the edge network need not lookup a key each time he sends/receives a packet. Such protocols are also particularly useful for *broadcast communications*.

*Detecting traffic discrimination.* Recently, there have been cases of ISPs that degrade performance for certain classes of unwanted traffic like Skype [37] or BitTorrent [1]. Thus, we prefer protocols that can be adapted to determine if a path is selectively dropping specific classes of traffic.

*Symmetric vs asymmetric paths.* Our PQM protocols are designed to ensure that Alice raises an alarm when the performance of the forward path (from Alice to Bob) degrades unacceptably. However, consider a situation where the performance of the forward path is acceptable, but Alice still raises an alarm because the adversary was tampering with messages sent on the reverse path (from Bob back to Alice). Our protocols *do not* protect against such situations; indeed, to design PQM protocols that give this guarantee, we would either need to assume that source and destination have an out-of-band communication channel that cannot be attacked by the adversary, or consider running PQM protocols over multiple alternate paths. Notice that when the forward path and reverse paths are identical, *i.e.,* symmetric paths, Eve has no incentive to drop acknowledgments and reports; doing this simply makes the path she occupies look worse. In contrast, with asymmetric paths, an adversary occupying only the reverse path may have an incentive to drop acknowledgments and reports, perhaps to confuse the source into thinking that the forward path is faulty.

However, some of our PQM protocols contain clues that Alice can often use to distinguish between situations where the forward path is actually faulty, and when an adversary on the reverse path is simply dropping reports.

*Monotonicity.* We say that that a protocol is *monotone* if Helen cannot trick the source into detecting faults on the data path simply by adding packets to the path. To see why this important, consider an adversary, Helen, that does *not* occupy a node on the data path and thus cannot drop or delay packets, but can *inject* packets onto the data path. Helen might have an incentive to trick Alice into raising an alarm this in order to force the Alice to switch her traffic to a different path. In practice, no protocol is completely monotone, since Helen can always cause a denial-of-service attack by flooding the path with nonsense packets. However, we typically want to avoid protocols where Helen can trick the source into detecting a failure (when all packets were

delivered) because of additional packet injections.

## 7.2 Evaluating the tradeoffs

We now discuss how each of our three protocols fits into the tradeoff space we described above. This discussion is summarized in Table 3.

*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.

*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* [7], 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.

*Stealth Probing.* Stealth Probing [7] is a network layer protocol that provides statistically-secure path-quality monitoring (satisfying Definition 2.1) by designating specific packets as 'probes' that must be ack'd by the destination, and then masking the choice of probe by encrypting and authenticating all traffic using IPsec. This protocol shares many of the traits of our symmetric secure sampling protocol. However, it incurs the extra overhead of encrypting all traffic, and is probably best suited when confidentiality is required in addition to PQM in the peer setting.

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 [37] 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.[12]

# 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, to operate in an adversarial setting as needed. Another possibility is to use our protocols with publicly known keys, but

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

[12]Of course, if packets are encrypted but not padded to a fix constant length, an adversary can still selectively mistreat certain packets based on their lengths. Furthermore, encryption does not prevent the adversary from using timing attacks to discriminate between packets, see *e.g.,* [45].

| | Secure Sampling | | Secure |
|---|---|---|---|
| | Sym | Asym | Sketching |
| Storage/Communication[11] | 90KB | 240–840KB | 0.2–2.3KB |
| Peer setting | ✓ | | ✓ |
| Client-server setting | | ✓ | |
| No clock sync | ✓ | coarse | ✓ |
| Estimates delay | ✓ | ✓ | |
| Fast feedback | ✓ | | |
| Monotonicity | ✓ | ✓ | |

**Table 3: Tradeoff space for our protocols.**

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. 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] R. Ahlswede and A. Winter. Strong converse for identification via quantum channels. *IEEE Trans. IT*, 48(3):569–579, 2002.

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

[6] D. Angulin and L. Valiant. Fast probabilistic algorithms for hamiltonian circuits and matchings. *J. of Computer and System Sciences*, 19:155–193, 1979.

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

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

[9] F. Bergadano, D. Cavagnino, and B. Crispo. Chained stream authentication. In *Workshop on Selected Areas in Cryptography*, pages 144–157, 2000.

[10] D. J. Bernstein. Polynomial evaluation and message authentication. Technical report, http://cr.yp.to/ Document ID: b1ef3f2d385a926123e1517392e20f8c., October 2007.

[11] B. Briscoe. FLAMeS, Tech. Rep., BT Research, 2000. `http://www.labs.bt.com/people/briscorj/papers.html`.

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

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

[14] S. Cheung. An efficient message authentication scheme for link state routing. In *Annual Computer Security Applications Conference*, pages 90–98, 1997.

[15] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1), 2005.

[16] J. Daemen and V. Rijmen. A new MAC construction ALRED and a specific instance ALPHA-MAC. In *FSE: Fast Software Encryption*, volume 3557, pages 1–17. Springer, 2005.

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

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

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

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

[21] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path quality monitoring in the presence of adversaries. In *SIGMETRICS*, June 2008.

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

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

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

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

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

[27] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. of the ACM*, 53(3):307–323, 2006.

[28] H. Krawczyk, M. Bellare, and R. Canetti. HMAC : Keyed-Hashing for Message Authentication. RFC 2104, 1997.

[29] T. D. Krovetz. *Software-optimized universal hashing and message authentication*. PhD thesis, University of California, Davis, 2000.

[30] K. Levchenko. Chernoff bound. Online. www-cse.ucsd.edu/ klevchen/techniques/chernoff.pdf.

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

[32] D. A. McGrew and J. Viega. The security and performance of the galois/counter mode (GCM) of operation. In *INDOCRYPT*, pages 343–355. Springer-Verlag, 2004.

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

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

[35] 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.

[36] M. Motiwala, A. Bavier, and N. Feamster. Network troubleshooting: An in-band approach (poster). *NSDI*, 2007.

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

[38] V. Paxson. End-to-end Internet packet dynamics. *IEEE Trans. on Networking*, 7(3):277–292, 1999.

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

[40] 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.

[41] P. Rogaway. Formalizing human ignorance: Collision-resistant hashing without the keys. In *Vietcrypt*, volume 4341, pages 211–228. Springer, 2006.

[42] M. Roughan. Fundamental bounds on the accuracy of network performance measurements. In *ACM SIGMETRICS*, 2005.

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

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

[45] D. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and SSH timing attacks. In *10th USENIX Security Symposium*, 2001.

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

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

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

[49] M. N. Wegman and J. L. Carter. New hash functions and their use in authentication and set equality. *JCSS*, 22:265–279, 1981.

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

# APPENDIX

# A. FAST CRYPTOGRAPHIC HASHING

To support multi-Gbit/sec packet streams we suggest a more efficient function for packet hashing. As we discussed in Section 3, in the PQM setting, the adversary is presented with an online problem: if Eve does not break the secret key within a small time interval until the key is refreshed, and based on the limited number of examples she sees during that interval, then she cannot compromise the measurement at all. (Indeed, in some of our protocols we voluntarily send the key in plaintext once the interval is over, see Section 4.2). Furthermore, in all of our protocols, cryptographic hash computation can be done after packet transmission and thus does not affect latency. Moreover, the hash computation on different packets can be arbitrarily parallelized and pipelined.

For general hashing of variable-length packets (lengths upto, say 1500B) with a PRF, we suggest a construction based on $\varepsilon_h$-almost universal hash functions, as discussed in [49] [29, Sec. 2.8.3] [10, 32]. Namely, for an interval key $k_u = (\kappa_1, \kappa_2)$ set

$$h_{k_u}(x) = E_{\kappa_1}(g_{\kappa_2}(x)) \tag{17}$$

where $E$ is a block cipher taking $n$-bit inputs to $n$ bit outputs (*e.g.,* AES), and $g$ is an $\varepsilon_g$-almost universal hash function producing $n$-bit outputs (as defined in equation (1) of Section 3). See *e.g.,* [10] for a nice survey of various $\varepsilon_h$-almost universal hash functions that can be used with this construction. [13] For long packets (length $\gg n$-bits), the performance of this hash function is dominated by the performance of the universal hash function $g$, which can extremely fast. For shorter packets (of lengths $\approx n$-bits), performance is limited by the block cipher; fortunately, this construction amounts to using a single invocation of the block cipher (*c.f.,* with traditional PRFs that require $\ell/n$ invocations of the block-cipher for packets of length $\ell$-bits.) Performance can be further improved by the replacing full-fledged block-cipher like AES with a weaker block cipher such as DES or with a small number of rounds of AES [16]; it would still require enormous resources to break the security of the PRF within the time limit ( 100ms) imposed in our online setting.

Unfortunately, most of the literature focuses on the construction of fast universal-hash based *MACs*, rather than PRFs. Thus, below, we shall show that the PRF construction presented in (17) is a secure PRF as long as the $\varepsilon_g$ universal hash function $g$ used in the construction has

$$\varepsilon_g \leq 2^{-(2\log_2 q + k)} \tag{18}$$

and the PRF is used no more than $q$ times before the interval key is refreshed. In our setting, $q = O(T)$, where $T$ is the number of packets per interval[14], and $k$ is the security parameter. In our online setting, $k$ need not be very large. As an example, putting $q = 10T$ and $T = 10^7$ and requiring $k > 32$, it follows that we require $\varepsilon_g \leq 2^{-86}$. Suppose now we use GHASH [32] with block lengths $n$ as the universal hash function used in the construction. Then for packets of

[13]The original universal-hash-based MACs require an an extra nonce $r$ which must be unique for each invocation, and define $h_k(x, r) = E_{\kappa_1}(r) + g_{\kappa_2}(m)$ or $h_k(x, r) = E_{\kappa_1}(r) \oplus g_{\kappa_2}(m)$. Because, in our application, we need only hash a small number of packets $\approx T = 10^7$ before changing the interval key, this nonce is not required.

length upto 1500B bytes, for GHASH have $\varepsilon_g = 1500/m2^{-n}$ where $m$ is the block length and $n$ is the length of the output. Thus, we find it suffices to run GHASH with output length $n = 96$ and block lengths of $m = 128$ bits. (A general rule of thumb is that hashing is faster with smaller block lengths. Note also that in Section 5.4.4, we show that for the 'secure sketch' protocol, it actually suffices to use GHASH with block lengths of about $m = 64$ bits!)

We now obtain (18). We say that a function $h$ is $(q, \varepsilon)$-secure PRF if an algorithm, given an oracle for a function $F$ has advantage at most $\varepsilon$ in distinguishing if $F$ is either (1) $h$ keyed with some randomly chosen secret key $k_u$, or (2) a truly random function with the appropriate domain and range [22]. To show that our construction in (17) is a secure PRF, we prove the following theorem:

THEOREM A.1. *The function $h_\kappa(x) = f(g_\kappa(x))$ is a $(q, q^2\varepsilon_g)$-secure PRF if $f$ is a* truly random function *and $g$ is an $\varepsilon_g$-almost universal hash function keyed with randomly-chosen key $\kappa$.*

Then the security of our construction follows if we assume that the block cipher $E$ produces (pseudorandom) outputs that are indistinguishable from the outputs of a truly random function. Thus, we require $q^2\varepsilon_g \leq 2^{-k}$, where $k$ is the security parameter for the PRF, and (18) follows.

PROOF PROOF OF THEOREM A.1. The algorithm makes $q$ queries $x_1, ..., x_q$ to the oracle for $F$; without loss of generality, assume these are all distinct. (If the adversary repeats a query, he can compute the answer without consulting the oracle.) Let $E_{\text{distinct}}$ be the event that $\kappa$ was chosen such that that $g_\kappa(x_1), ..., g_\kappa(x_q)$ are all distinct. Since $f$ is a truly random function taking in distinct inputs, it follows that if $E_{\text{distinct}}$ is true, the adversary has no advantage in distinguishing between $h$ and a truly random function. Thus, the adversary's distinguishing advantage is at most $\Pr[\neg E_{\text{distinct}}]$. Now, for every distinct $x_1, ..., x_q$, we have $\Pr[\neg E_{\text{distinct}}] = \binom{q}{2}\varepsilon_g \leq q^2\varepsilon_g$ where $\varepsilon_g$ is the collision probability of $g$ (taken over the choice of $\kappa$) as in equation (1). The theorem follows. □

# B. INTERVAL SYNCHRONIZATION

## B.1 Symmetric-key protocols

In our secure sketch (Section 5) and symmetric secure sampling protocols (Section 4.1), we assume that Alice and Bob agree on the set of packets belonging to a particular interval, and process these packets using the same interval key $k_u$ (and, in Section 5, map the same set of packets to the same sketch.) For these protocols, the best way to achieve this is to have Alice send Bob a special 'Interval End' message each time she ends an interval and begins a new one. The 'Interval End' message should contain the interval number $u$, and be authenticated with a MAC keyed with (some portion of) the master secret key. When Bob receive this packet, he knows he should derive a fresh interval key (and, in Section 5, a fresh sketch). This approach does not make any synchronization assumptions about Alice and Bob's local clocks; it also works even if the path between Alice and Bob is subject to variable latency.

**The effect of packet reordering on interval synchronization.** Of course, in the benign case, out-of-order arrival could cause packets in an interval $u$ to arrive after the interval marker packet for $u$ (and thus be interpreted by Bob as part of interval $u + 1$). Fortunately, out-of-order arrival should not cause any false alarms as long as the number of packets arriving out of order before the interval marker is a small fraction of $\alpha T$, where $\alpha$ the false-alarm threshold. (Note that because we focus on PQM protocols that operate at the network layer, at this layer TCP retransmissions do not look like out-of-order packets.)

Indeed, out-of-order arrive limits the choice of $\alpha T$. To see how, consider an ordered stream of packets transmitted by a sender (*e.g.,* 1,2,3,4,5,6,7,8). Let a "reordered packet" be some packet that arrives at the receiver later than the packets after it in the ordered stream sent by the sender (*e.g.,* . in received stream 1,2,4,5,6,7,3,8, packet 3 is the reordered packet). Then, define the packet lag as the number of packets that were sent by the sender after the reordered packet, but were received at the receiver earlier than the reordered packet itself (*e.g.,* in received stream 1,2,4,5,6,7,3,8, packet 3 is the reordered packet and packet lag is 4).

To ensure natural packet reordering on the link does not cause a loss of interval synchronization between the sender and receiver, a good rule of thumb is to ensure that $\alpha T \geq 99^{th}$ percentile of packet lag. The packet lag depends on the the class of packets monitored by the PQM protocol. For instance:

- If the PQM protocol is in setting where no load balancing is used, (*i.e.,* packets sent by Alice to Bob are sent over a single physical path through the network, rather than split over multiple paths) then packet lag is typically very small, *i.e.,* about 10's of packet [38, Sec. III.A]. Thus, ensuring $\alpha T \geq 100$ is sufficient in this case.
- If the PQM protocol is used to monitor a single "layer-3 flow", *i.e.,* a set of IP packets with same (Source IP, Destination IP, Source Port, Destination Port, Protocol Numbuer), then we assume that packet lag is less than 128 packets. (This is the assumption made in IPSec). Thus, it suffices to take $\alpha T > 1280$.
- If the PQM protocol simultaneously monitors multiple layer 3 flows, then packet lag can be quite high. This is because different flows may be routed on different paths through the network; if there is a significant time delay between the different paths used by the different flows, then packet lag can be very high. The best way to determine packet lag in this setting is to measure it directly; however, we conjecture that even if there is a 10ms difference between the "fast path" used by one group of flows and the "slow path" used by another group of flows, for 1 Gbps flow of traffic, packet lag should be on the order of $10^9$ bps / 64 bytes/packet * .01 sec = $1.6x10^5$ packets, so we can use $\alpha T > 1.6x10^6$ .

Also, if the link has many out-of-order packets even in the

[14] In the secure sampling protocol, $q$, the number of queries made to the packet-hashing PRF is the sum of the number of packets that Alice sends and the number of packets that Bob receives. Notice that an adversary can exceed the bound of $q = O(T)$ by adding many packets to path, and potentially use the information it sees (*i.e.,* the ACKs) to learn how to break the PRF. To avoid such problems, we suggest that Bob counts the number of packets he receives in an interval, and stops acknowledging them once more than $q$ packets have been received.

benign case, we can enforce interval synchronization by marking packets with a single bit denoting the parity of the interval number (note that if the adversary tampers with this mark, she only increases the likelihood that Alice will raise an alarm).[15]

Finally, notice that if Eve drops or delays the marker packet for interval $u$, then she only increases the changes that Alice raises an alarm (since doing is equivalent to adding many packets to interval $u$ and dropping many packets in interval $u+1$).

## B.2 Asymmetric-key protocols

Our asymmetric secure sampling protocols (Section 4.2) use a different approach for interval synchronization. Here, the end of the interval is determined when the server sends out the 'salt release message'. Thus, there is no need to have the client send the server an 'interval marker packet'. We do, however, require Alice to be coarsely synchronized to Bob's clock, so that an adversary cannot replay old salt release messages (and use the old salt to form ACKs that trick the client into accepting an interval for which she should have raised an alarm).

In settings where the sender and receiver do not share a clock, the following simple protocol can be used to securely synchronize Alice's clock to Bob's clock to within 1.5 round trip times (RTT) (*e.g.,* $\tau = 150$ ms). Notably, this protocol does not require either Alice or Bob to keep any state beyond their keys and local clocks. The protocol also does not require Alice and Bob to trust one another, and does not affect Alice's global clock that is used when interacting with other parties.

**Simple synchronization protocol.** Suppose Alice has some local secret key $k_A$ (she does not need to shared this key with anyone).

1. At time $t_A$ (on Alice's clock) Alice sends Bob the message $\mathrm{MAC}_{k_A}(t_A)$.
2. Bob receives this message at time $t_B$ (on Bob's clock) and responds with digitally signed message $\xi = \mathrm{Sign}_{SK_B}(t_B, \mathrm{MAC}_{k_A}(t_A))$.[16]
3. Alice accepts Bob's message $\xi$ if $\mathrm{Verify}_{PK_B}(\xi)$ returns $(t_B, \mathrm{MAC}_{k_A}(t_A))$, the MAC is correct, and Alice's current local time $t'_A$ fulfills $t'_A < t_A + \tau$. If Alice accepted Bob's message, she computes $\Delta_B = \tau - t'_A$, and from now on, whenever interacting with Bob she offsets her clock by a factor of $\Delta_B$.

If, after many attempts, Alice fails to receive a valid response to her synchronization message, then she decides to raise an alarm. After Alice accepts, her local clock (after being offset by $\Delta_B$) is within $\tau$ seconds from Bob's regardless of Eve's actions. Indeed, a sufficient condition is that any accepted message $\xi$ was sent by Bob when his local time was $t'_B$ and Alice's local time was after $t'_A$. Violating either of these would contradict the security of the digital signature and MAC schemes.

[15] As packet marking is highly undesirable, it might be better to use alternative options such as increasing the interval length. Having a secret random interval starting time also seems to help to some extent to ensure that out-of-order packets do not effect the benign setting more than the adversarial setting, by ensuring the adversary cannot give preferential treatment to packets close to the interval boundary. We have not fully analyzed this solution.

[16] While computing and verifying digital signatures typically takes on the order of 3ms, and is thus insignificant as compared to the 150ms interval consider here. Furthermore, this time delay is constant and known and can be subtracted from $\Delta_B$.

# C. SECURE SAMPLING NEEDS PRFS

To give an example of why non-cryptographic hash functions are not insufficient in our sampling protocols, suppose that the Probe function of equation (3) was implemented using a CRC keyed with a secret modulus, as in [19], instead of a PRF. Approximate the CRC function as $h_k(x) = x \bmod k$, and consider the following attack: Eve starts by observing the interactions on the channel, and records the list of packets that were not acknowledged. 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 the estimate $V$ well below the true failure rate. This attack is possible because the CRC does not use its "secret key" in a "cryptographically-strong" manner.

# D. PREHASHING PACKETS

As discussed in Section 5.4.4, arguably the most expensive part of our sketching protocol is the computation of the per-packet hash. We now show how to reduce the cost of this computation by (1) first mapping packets from from $U$ to a short $n_1$-bit string using the efficient $\varepsilon_g$-almost universal hash function, and (2) then using a PRF (or 4-wise independent) hash to map from $n_1$-bit to the sketch. Our approached is based on that of Thorup and Zhang [48].

**Preliminaries.** Recall that $U$ is the universe of all possible packets, and $\mathbf{v}$ is the characteristic vector of the stream of packets. Let $g : U \to \{0,1\}^{n_1}$ be an $\varepsilon_g$-almost universal hash function, as defined in Section 3. The hash function $g$ maps the packet stream containing elements in $U$ to a new 'intermediate' stream of $n_1$-bit strings. Now, we let $\mathbf{u}$ be an 'intermediate vector' which is the characteristic vector of this new stream of $n_1$-bit strings. And finally, recall that $\mathbf{w}$ is the sketch vector of length $N$.

Thus, our approach amounts to using the $\varepsilon_g$-almost universal hash $g$ to hash $\mathbf{v}$ the 'intermediate vector' $\mathbf{u}$, and then using a a second-moment estimation scheme to hash $\mathbf{u}$ down to the sketch $\mathbf{w}$. Thus, the second-moment estimation scheme estimates the second moment of $\mathbf{u}$, rather than the real characteristic vector $\mathbf{v}$! We now show that, if $\varepsilon_g$ is sufficiently small, this does very little damage, since $\|\mathbf{u}\|_2 \approx \|\mathbf{v}\|_2$.

THEOREM D.1. *Given a vector $\mathbf{v} \in 2^{|U|}$ and $\mathbf{u} \in \mathbb{R}^{2^{n_1}}$. Then if $g : U \to \{0,1\}^n$ is an $\varepsilon_g$-almost 2-wise independent hash function per equation (1), is used to map $\mathbf{v}$ to $\mathbf{u}$ according to the algorithm $u_{g(x)} += v_x$ (*i.e., *$\forall\, x \in \mathbf{v}$ the $g(x)^t h$ counter in $\mathbf{u}$ is incremented with value $v_x$) then*

$$\Pr\left[\,|\|\mathbf{u}\|_2 - \|\mathbf{v}\|_2| > \delta_1 \|\mathbf{v}\|_2\right] < \delta_2 \tag{19}$$

*as long as*

$$|\mathbf{v}|_1 > \frac{\delta_1 \delta_2}{\varepsilon_g} \tag{20}$$

Recall that $|\mathbf{v}|_1 = A + D$. Returning the proof of Theorem 5.1, for $(\alpha, \beta, \delta)$-secure PQM we would like (19) to hold

particularly when $D = \alpha T$ and $D = \beta T$, with $\delta_1 \ll \varepsilon = \frac{\beta-\alpha}{\alpha+\beta}$. To be more conservative, we will take $|\mathbf{v}|_1 = T$, and $\delta_1 = \frac{\varepsilon}{10}$. We'll also set $\delta_2 = \frac{\delta}{100}$. Then $(\alpha, \beta, \delta)$-secure PQM require the hash function $g$ to have $\varepsilon_g$ as :

$$\varepsilon_g < \frac{\varepsilon\delta}{10^3 T} = \frac{\delta}{10^3 T}\frac{\beta-\alpha}{\alpha+\beta} \tag{21}$$

Proof Proof of Theorem D.1. We start with the observation that

$$\|\mathbf{u}\|_2^2 = \sum_{g(a)=g(b)} v_a v_b \tag{22}$$

$$= \sum_a v_a^2 + \sum_{a\neq b, g(a)=g(b)} v_a v_b \tag{23}$$

$$= \|\mathbf{v}\|_2^2 + \sum_{a\neq b} v_a v_b Y_{a,b} \tag{24}$$

where we define the random variable $Y_{a,b}$ as

$$Y_{a,b} = \begin{cases} 1 & \text{if } g(a) = g(b), a \neq b, \\ 0 & \text{else.} \end{cases} \tag{25}$$

and from (24) we take the expectation over the randomness in $g$ and find that

$$\mathbb{E}[\,|\,\|\mathbf{u}\|_2^2 - \|\mathbf{v}\|_2^2\,|\,] \leq \sum_{a,b} |v_a v_b| \mathbb{E}[|Y_{a,b}|] \tag{26}$$

$$\leq \sum_{a,b} |v_a v_b| (\varepsilon_g) \tag{27}$$

$$= (|\mathbf{v}|_1^2 - \|\mathbf{v}\|_2^2)(\varepsilon_g) \tag{28}$$

where the first inequality follows from (24), the second inequality follows because per equation (1) the collision probability of $g$ is $\varepsilon_g$.

Now, we would like to ensure that $\|\mathbf{u}\|_2$ provides a good estimate of $\|\mathbf{v}\|_2$. That is, we would like to satisfy (19). Using Markov's inequality, we have

$$\Pr\left[\left|\|\mathbf{u}\|_2^2 - \|\mathbf{v}\|_2^2\right| > \delta_1 \|\mathbf{v}\|_2^2\right] \leq \frac{\mathbb{E}[\,|\,\|\mathbf{u}\|_2^2 - \|\mathbf{v}\|_2^2\,|\,]}{\delta \|\mathbf{v}^2\|_2} \tag{29}$$

$$\leq \frac{(|\mathbf{v}|_1^2 - \|\mathbf{v}\|_2^2)}{\|\mathbf{v}\|_2^2}\frac{\varepsilon_g}{\delta_1} \tag{30}$$

$$\leq |\mathbf{v}|_1 \frac{\varepsilon_g}{\delta_1} \tag{31}$$

And rearranging the last inequality we know that (19) holds as long as (20) holds, which completes the proof. □

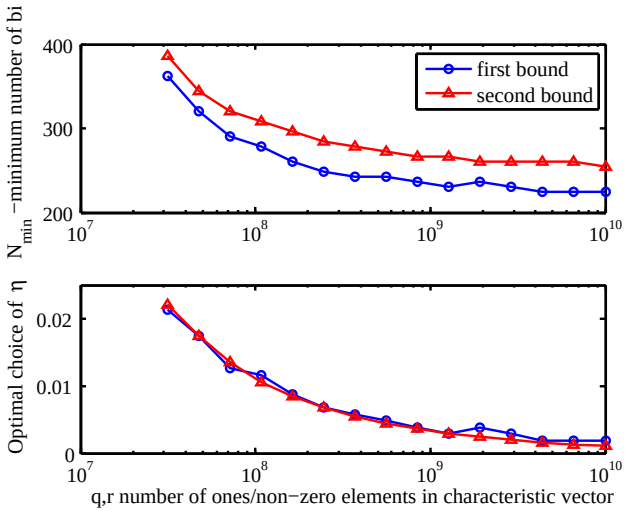


**Figure 6: An example of how to use Theorem E.4.**

# E. SECOND-MOMENT ESTIMATION WITH CCF

We start by restating a result of Thorup and Zhang from [48].

THEOREM E.1. *For any characteristic vector $\mathbf{v} \in \mathbb{R}^{|U|}$, if we construct sketch $\mathbf{w} \in \mathbb{Z}^N$ according to the CCF-algorithm with two independent 4-universal hash functions $h : U \to [N]$ and $s : U \to \{-1, 1\}$ as $w_{h(a)} += s(a)v_a$ then we have*

$$\mathbb{E}[\|\mathbf{w}\|_2^2] = \|\mathbf{v}\|_2^2 \tag{32}$$

$$\mathbb{VAR}[\|\mathbf{w}\|_2^2] = \tfrac{2}{N}(\|\mathbf{v}\|_2^4 - \|\mathbf{v}\|_4^4) \tag{33}$$

## E.1 CCF with 4-wise independent hashes

To show that the number of bins required for CCF with a 4-wise independent hash is $N > \frac{2}{\varepsilon^2\delta}$ we prove the following theorem.

THEOREM E.2. *For any vector $\mathbf{v} \in \mathbb{R}^U$ sketched according to the CCF-algorithm with a 4-wise independent hash to obtain $\mathbf{w} \in \mathbb{Z}^N$, then for any $\varepsilon \in (0, 1)$ we have*

- *If $\|\mathbf{v}\|_2^2 \le \alpha T$ and $\alpha T > 1$ then*

$$\Pr\left[\|\mathbf{w}\|_2^2 < (1+\varepsilon)\alpha T\right] \le \tfrac{2}{N\varepsilon^2}(1 - \tfrac{1}{\alpha T}) \tag{34}$$

- *If $\|\mathbf{v}\|_2^2 \ge \beta T$ and $\beta T > 1$ then*

$$\Pr\left[\|\mathbf{w}\|_2^2 > (1-\varepsilon)\beta T\right] \le \tfrac{2}{N\varepsilon^2} \tag{35}$$

Then, recall that we set our threshold as $\Gamma = (1-\varepsilon)\beta T = (1+\varepsilon)\alpha T$ for $\varepsilon = \frac{\beta-\alpha}{\beta+\alpha}$. Following the approach in the proof of Theorem 5.1, we can show that the scheme is $(\alpha, \beta, \delta)$-secure PQM protocol per Definition 2.1 if if we take a sketch with at least $N > \frac{2}{\delta\varepsilon^2} = \frac{2}{\delta}(\frac{\beta+\alpha}{\beta-\alpha})^2$ bins.

PROOF PROOF OF THEOREM E.2. We start with the first item and write:

$$\Pr\left[\|\mathbf{w}\|_2^2 < (1+\varepsilon)\alpha T\right] \le \frac{\mathbb{VAR}[\|\mathbf{w}\|_2^2]}{\left((1+\varepsilon)\alpha T - \mathbb{E}[\|\mathbf{w}\|_2^2]\right)^2} \tag{36}$$

$$\le \frac{2}{N}\frac{(\|\mathbf{v}\|_2^4 - \|\mathbf{v}\|_4^4)}{\left((1+\varepsilon)\alpha T - \|\mathbf{v}\|_2^2\right)^2} \tag{37}$$

$$\le \frac{2}{N}\frac{(\|\mathbf{v}\|_2^4 - \|\mathbf{v}\|_2^2)}{\left((1+\varepsilon)\alpha T - \|\mathbf{v}\|_2^2\right)^2} \tag{38}$$

$$\le \frac{2}{N}\frac{(\alpha T)^2 - \alpha T}{(\varepsilon\alpha T)^2} \tag{39}$$

$$= \frac{2}{N\varepsilon^2}(1 - \tfrac{1}{\alpha T}) \tag{40}$$

where the first inequality follows from the Chebyshev bound, the first equality follows from applying Theorem E.1, and second inequality follows because for any $\mathbf{v} \in \mathbb{Z}^N$ we have $\|\mathbf{v}\|_2^2 \le \|\mathbf{v}\|_4^4$ (this holds because $\mathbf{v}$ has integer valued entries), and the third inequality follows from the fact that $\|\mathbf{v}\|_2^2 \le \alpha T$ and $\alpha T > 1$.

Next, consider the second item and write:

$$\Pr\left[\|\mathbf{w}\|_2^2 < (1-\varepsilon)\beta T\right] \le \frac{\mathbb{VAR}[\|\mathbf{w}\|_2^2]}{\left(\mathbb{E}[\|\mathbf{w}\|_2^2] - (1-\varepsilon)\beta T\right)^2} \tag{41}$$

$$\le \frac{2}{N}\frac{(\|\mathbf{v}\|_2^4 - \|\mathbf{v}\|_4^4)}{\left(\mathbb{E}[\|\mathbf{w}\|_2^2] - (1-\varepsilon)\beta T\right)^2} \tag{42}$$

$$\le \frac{2}{N}\frac{\|\mathbf{v}\|_2^4}{\left(\|\mathbf{v}\|_2^2 - (1-\varepsilon)\beta T\right)^2} \tag{43}$$

where again the first inequality follows from the Chebyshev bound, the first equality follows from applying Theorem E.1, and second inequality follows because $\|\mathbf{v}\|_4^4 \ge 0$ since $\beta T > 1$. To bound (43), we use the following claim.

CLAIM E.3. *For any $a > 0$, and function $f(x) = \frac{x}{x-a}$ decreases with $x$ when $x > a$.*

PROOF. Follows because $\frac{df}{dx} = -\frac{a}{(x-a)^2} < 0$ and there are no singularities in $f(x)$ for all $0 < a < x$. □

To apply Claim E.3, let $x = \|\mathbf{v}\|_2^2$ and $a = (1-\varepsilon)\beta T$ and recall from the statement of the theorem that $x \ge \beta T > a$. From Claim E.3 we note that (43) takes on its largest value when $x = \beta T$, so we can write (43) as

$$\Pr\left[\|\mathbf{w}\|_2^2 < (1-\varepsilon)\beta T\right] \le \frac{2}{N}\frac{(\beta T)^2}{\left(\beta T - (1-\varepsilon)\beta T\right)^2} \tag{44}$$

$$= \tfrac{2}{N\varepsilon^2} \tag{45}$$

which completes the proof.

## E.2 CCF with PRFs

First we prove a precise version of Theorem E.4.

THEOREM E.4. *For any vector $\mathbf{v} \in \mathbb{Z}^U$, choosing the $N \times U$ matrix $S$ uniformly from $\mathcal{S}_{\text{CCF}}$ and setting $\mathbf{w} = S\mathbf{v}$, we have that for all $\varepsilon \in [0,1)$ and all $q, r > N$*

1. *If $\mathbf{v} \in \{-1,0,1\}^U$, and $\|v\|_2^2 \le q$, then for $\eta \in [0, \frac{1}{2}\sqrt{\varepsilon^2+10\varepsilon+9} - \frac{1}{2}(\varepsilon+3))$ and $y \doteq \frac{(1+\varepsilon)(1-\eta)}{(1+\eta)^2} - 1$ :*

$$\Pr\left[\|\mathbf{w}\|_2^2 > (1+\varepsilon)q\right] \le 2Ne^{-\frac{\eta^2 q}{3N}} + e^{-\frac{N}{2}(y^2/2 - y^3/3)} \tag{46}$$

2. *If the number of* non-zero *entries in $\mathbf{v}$ is $r$, then for $\eta \in \left(0, \frac{1}{2-\varepsilon}(3-2\varepsilon-\sqrt{5\varepsilon^2-14\varepsilon+9})\right)$ and $y \doteq \frac{(1-\eta)^2}{1+\eta}(1-\frac{\varepsilon}{2}) - (1-\varepsilon)$ it follows that*

$$\Pr\left[\|\mathbf{w}\|_2^2 < (1-\varepsilon)r\right] \le 2Ne^{-\frac{\eta^2 r}{3N}} + e^{-N\frac{\varepsilon}{3(1+\eta)}y} \tag{47}$$

For a fixed $\varepsilon$, suppose we want to choose values for $q, r, N$ that ensure that both the first item and the second item occur with probability at most $\delta$. To use Theorem E.4, we need to choose a value of $\eta$ (within the appropriate range) and plug it into (46)-(47) to obtain $q, r$ and $N$. However, the first term $(2Ne^{-\frac{\eta^2 q}{3N}})$ in equations (46)-(47), *increases* in $\eta$, which the second term *decreases* in $\eta$. Thus, finding an optimal choice of $\eta$ the minimizes $N$ for a given choice of $q, r$ requires trading off between these two terms. This optimization can be messy, and so we do it in MATLAB. In Figure D, we set $\varepsilon = \frac{1}{3}$ and for $q = r$ we require that both the first item and the second item occur with probability at most $\delta = \frac{1}{100}$. For each value of $q$, we show the choice of $\eta$ that minimizes $N$ for both the first item and the second item in Theorem E.4.

PROOF PROOF OF THEOREM E.4. The main observation we make is that, with high probability, the $\pm 1$ entries of $\mathbf{v}$ are distributed evenly among the coordinates of $\mathbf{w}$. Conditioned on this happening, we can then apply the analysis of Achlioptas [3].

**Definitions.** We need the following definitions.

- We write $v_x$ for the $x^{th}$ element in $\mathbf{v}$.
- Define for $i \in [N]$ the set $Q_i = \{x \in U \mid h(x) = i\}$ where $h$ is the pseudorandom hash function.
- Define $D_i$ as the number of non-zero entries in $\mathbf{v}$ that hash to the $i^{th}$ bin the sketch $\mathbf{w}$. That is $D_i = |\{v_x | v_x \neq 0, x \in Q_i\}|$.
- Define $Y_x$ as an unbiased $\pm 1$ random variable for each $x \in U$.

Our proof proceeds as follows. We first obtain a bound on $D_i$ for each $i$. When then use the bounds on $D_i$ to prove the first item (46), and then use them to prove the second item (47).

**Bounding $D_i$.** Let $E_i$ denote the event that $\exists i \in [N]$ such that $D_i > (1+\eta)q/N$ or $D_i < (1-\eta)q/N$. Then, for $\eta \in [0,1)$, we have that

$$\Pr[E_1] \le N\left(\Pr[D_i > (1+\eta)\tfrac{q}{N}] + \Pr[D_i < (1-\eta)\tfrac{q}{N}]\right) \le N\left(e^{-\frac{\eta^2}{3}\frac{q}{N}} + e^{-\frac{\eta^2}{2}\frac{q}{N}}\right) \tag{48}$$

which is a straightforward application of a union bound followed by the Chernoff bound.[17]

**Bounding the first item.** Now we condition on $\neg E_1$. Let $\gamma = \frac{1+\varepsilon}{(1+\eta)^2}$ and write:

$$\begin{aligned}\Pr[\|\mathbf{w}\|_2^2 > (1+\varepsilon)q \mid \neg E_1] &= \Pr[\sum_{i=1}^N D_i^2 \left(\tfrac{1}{D_i}\sum_{x\in Q_i} Y_x v_x\right)^2 > (1+\varepsilon)q \mid \neg E_1] \\ &= \Pr[\sum_{i=1}^N \left(\tfrac{1}{D_i}\sum_{x\in Q_i} Y_x v_x\right)^2 > \gamma\tfrac{N^2}{q} \mid \neg E_1]\end{aligned}$$

[17] We use the following Chernoff bounds. Let $X_i$ be i.i.d indicator variables with mean $\mu$, and let

$$\Pr\left[\sum_{i=1}^n X_i \le (1-\gamma)N\mu\right] \le e^{-\gamma^2 N\mu/C_1} \tag{49}$$

$$\Pr\left[\sum_{i=1}^n X_i \ge (1+\gamma)N\mu\right] \le e^{-\gamma^2 N\mu/C_2} \tag{50}$$

If $0 < \gamma < 1$ then [6, Fact 4] gives $C_1 = 2$ and $C_2 = 3$. If $0 < \gamma < \frac{1}{2}$ then [4, Thm. 19] gives $C_1 = C_2 = 2\ln 2$.

where first equality comes from expanding $\mathbf{w}$ as $\mathcal{S}\mathbf{v}$ and then multiplying by $\frac{D_i}{D_i}$, and the second equality follows from the fact that conditioning on $\neg E_i$ implies that $D_i \leq (1+\eta)q/N$. Next, set $\mathbf{Y}_i$ to be the vector of all $Y_x$ for each $v_x \in -1, 1, x \in Q_i$. Set $\mathbf{u}_i$ the vector with entries $\frac{v_x}{\sqrt{D_i}}$ for each $v_x \in \{-1, 1\}, x \in Q_i$. Notice that both $\mathbf{Y_i}$ and $\mathbf{u_i}$ have length $D_i$, and $\|\mathbf{u}_i\|_2^2 = 1$ so $\mathbf{u_i}$ is a unit vector. Now we write

$$\begin{aligned}
&= \Pr[e^{t\sum_{i=1}^{N}\langle\frac{\mathbf{Y}_i}{\sqrt{D_i}},\mathbf{u}_i\rangle^2} > e^{t\gamma\frac{N^2}{q}} \mid \neg E_1] \\
&= \Pr[\sum_{i=1}^{N}\langle\frac{\mathbf{Y}_i}{\sqrt{D_i}},\mathbf{u}_i\rangle^2 > \gamma\frac{N^2}{q} \mid \neg E_1] \\
&\leq e^{-t\gamma\frac{N^2}{q}}\prod_{i=1}^{N}\mathbb{E}[e^{t\langle\frac{\mathbf{Y}_i}{\sqrt{D_i}},\mathbf{u}_i\rangle^2} \mid \neg E_1]
\end{aligned}$$

where the inequality follows from the Markov bound. Now we are ready to apply the result of Achlioptas. We restate equation (2) and Lemma 5.2 of [3] here, using our own terminology.

LEMMA E.5 (FROM [3]). *For $t \in [0, D_i/2]$, unit vector $\mathbf{u}_i$ (*i.e., *$\|\mathbf{u}_i\|_2^2 = 1$) and $\mathbf{Y}_i$ chosen uniformly from $\{1, -1\}^{D_i}$ we have that*

$$\mathbb{E}[e^{t\langle\frac{\mathbf{Y}_i}{\sqrt{D_i}},\mathbf{u}_i\rangle^2}] \leq \frac{1}{\sqrt{1-2t/D_i}} \tag{51}$$

$$\mathbb{E}[\langle\frac{\mathbf{Y}_i}{\sqrt{D_i}},\mathbf{u}_i\rangle^2] = \frac{1}{D_i} \tag{52}$$

$$\mathbb{E}[\langle\frac{\mathbf{Y}_i}{\sqrt{D_i}},\mathbf{u}_i\rangle^4] = \frac{3}{D_i^2} \tag{53}$$

Now, using Achlioptas's result in (51) we write

$$\begin{aligned}
\Pr[\|\mathbf{w}\|_2^2 > (1+\varepsilon)q \mid \neg E_1] &\leq e^{-t\gamma\frac{N^2}{q}}\prod_{i=1}^{N}\mathbb{E}[e^{t\langle\frac{\mathbf{Y}_i}{\sqrt{D_i}},\mathbf{u}_i\rangle^2} \mid \neg E_1] \\
&\leq e^{-t\gamma\frac{N^2}{q}}\prod_{i=1}^{N}\frac{1}{\sqrt{1-2t/D_i}} \\
&\leq e^{-t\gamma\frac{N^2}{q}}(1-\tfrac{2t}{(1-\eta)q/N})^{-\frac{N}{2}} \doteq v(t)
\end{aligned} \tag{54}$$

where the last inequality (54) follows from conditioning on $\neg E_i$ which implies that $(1-\eta)q/N < D_i$ for all $i \in [N]$. Note that result of Achlioptas in (51) to hold, we must have $0 \leq t < D_i/2 \leq \frac{(1+\eta)q}{2N}$ where the last inequality here follows from the fact that $\neg E_i$ implies that $D_i < (1+\eta)q/N$.

**Optimizing and bounding t.** Next, we optimize $v(t)$ in (54), by finding $t$ such that $\frac{dv(t)}{dt} = 0$.

$$\begin{aligned}
\tfrac{dv(t)}{dt} = -\tfrac{\gamma N^2}{q}v(t) + (-\tfrac{N}{2})(-\tfrac{2}{(1-\eta)q/N})(1-\tfrac{2t}{(1-\eta)q/N})^{-1}v(t) &= 0 \\
\tfrac{\gamma N^2}{q}(1-\tfrac{2t}{(1-\eta)q/N}) &= \tfrac{N^2}{(1-\eta)q} \\
t &= \tfrac{q}{2N}\left((1-\eta) - \tfrac{(1+\eta)^2}{1+\varepsilon}\right)
\end{aligned} \tag{55}$$

where the last equality uses the fact that $\gamma \doteq \frac{1+\varepsilon}{(1+\eta)^2}$. Now recall that for Achlioptas's result in (51) to hold, we need to ensure that $0 \leq t < \frac{(1+\eta)q}{2N}$. Using (55), we write

$$\begin{aligned}
0 &\leq t \\
0 &\leq \tfrac{q}{2N}\left((1-\eta) - \tfrac{(1+\eta)^2}{1+\varepsilon}\right) \\
\tfrac{(1+\eta)^2}{1+\varepsilon} &\leq (1-\eta) \\
\tfrac{(\eta^2+3\eta)}{1-\eta} &\leq \varepsilon
\end{aligned} \tag{56}$$

and we also need

$$\begin{aligned} t &< \tfrac{(1+\eta)q}{2N} \\ \tfrac{q}{2N}\left((1-\eta) - \tfrac{(1+\eta)^2}{1+\varepsilon}\right) &< \tfrac{(1+\eta)q}{2N} \\ -\tfrac{(1+\eta)^2}{1+\varepsilon} &< 2\eta \\ -\left(1 + \tfrac{(1+\eta)^2}{2\eta}\right) &< \varepsilon \end{aligned} \tag{57}$$

Now, (57) holds for any $\eta \in [0,1)$. But, we will need to ensure that our choice of $\eta \in [0,1)$ satisfies (56).

Returning now to (54), plug (55) into (54) to get

$$\Pr[\|\mathbf{w}\|_2^2 > (1+\varepsilon)q \mid \neg E_1] \leq \left(e^{-y}(1+y)\right)^{\frac{N}{2}} \tag{58}$$

where we define

$$y \doteq \frac{(1+\varepsilon)(1-\eta)}{(1+\eta)^2} - 1 \tag{59}$$

and solving inequality (56), we find that (58) holds as long as $\eta \in [0,1)$ satisfies

$$0 < \eta < \tfrac{1}{2}\left(\sqrt{\varepsilon^2 + 10\varepsilon + 9} - (\varepsilon + 3)\right) \tag{60}$$

Notice from (59) that the bound in (60) this implies that (58) holds for the region $y \in [0, \varepsilon)$. Now, Achlioptas observes that $e^{-y}(1+y) \leq e^{(-y^2/2 + y^3/3)}$ for any $y \in (0,1)$. Since for us $y \in (0,\varepsilon)$, and $\varepsilon < 1$ we finally have

$$\Pr[\|\mathbf{w}\|_2^2 > (1+\varepsilon)q \mid \neg E_1] \leq e^{-\frac{N}{2}(y^2/2 - y^3/3)} \tag{61}$$

which decays exponentially in $N$. $\square$

**Bounding the second item.** Let $r$ be the number of non-zero entries in $\mathbf{v}$. We will bound $\Pr[\|\mathbf{w}\|_2^2 < (1-\varepsilon)r]$. Define $E_1$ as before, only this time use $r$ instead of $q$. Again we condition on $\neg E_1$.

$$\begin{aligned} \Pr[\|\mathbf{w}\|_2^2 < (1-\varepsilon)r \mid \neg E_1] &= \Pr[\sum_{i=1}^{N} D_i^2 \left(\tfrac{1}{D_i}\sum_{x \in Q_i} Y_x v_x\right)^2 < (1-\varepsilon)r \mid \neg E_1] \\ &= \Pr[\sum_{i=1}^{N} \left(\tfrac{1}{D_i}\sum_{x \in Q_i} Y_x v_x\right)^2 < \tfrac{(1-\varepsilon)}{(1-\eta)^2}\tfrac{N^2}{r} \mid \neg E_1] \end{aligned}$$

where first equality comes from the expanding $\|\mathbf{w}\|_2^2$and then multiplying by $\frac{D_i}{D_i}$, and the second equality follows from the fact that conditioning on $\neg E_i$ implies that $(1-\eta)r/N < D_i$. Next, we let $c_i^2 = \sum_{x \in Q_i} \frac{v_x^2}{D_i}$. Now observe that $c_i^2 = \frac{1}{D_i}\sum_{x \in Q_i} v_x^2 \geq \frac{1}{D_i} D_i = 1$ since the entries of $v$ are integers ( and $D_i$ is the number of non-zero entries in $v$ that are in $Q_i$). We now multiply by $\frac{c_i}{c_i}$:

$$\begin{aligned} &= \Pr[\sum_{i=1}^{N} c_i^2 \left(\sum_{x \in Q_i} Y_x \tfrac{v_x}{D_i c_i}\right)^2 < \tfrac{(1-\varepsilon)}{(1-\eta)^2}\tfrac{N^2}{r} \mid \neg E_1] \\ &\leq \Pr[\sum_{i=1}^{N} \left(\sum_{x \in Q_i} Y_x \tfrac{v_x}{D_i c_i}\right)^2 < \tfrac{(1-\varepsilon)}{(1-\eta)^2}\tfrac{N^2}{r} \mid \neg E_1] \end{aligned}$$

where the inequality follows from the fact that $c_i^2 \geq 1$. We now set $\mathbf{Y}_i$ to be the vector of all $Y_x$ for each $v_x \neq 0, x \in Q_i$. Set $\mathbf{u}_i$ the vector with entries $\frac{v_x}{\sqrt{D_i c_i}}$ for each $v_x \neq 0, x \in Q_i$. Notice that both $\mathbf{Y}_i$ and $\mathbf{u}_i$ have length $D_i$, and that $\mathbf{u}_i$ is a unit vector, since $\|\mathbf{u}_i\|_2^2 = \frac{1}{D_i c_i^2}\sum_{x \in Q_i} v_x = \frac{c_i^2}{c_i^2} = 1$. We write

$$\begin{aligned} &= \Pr[\sum_{i=1}^{N} \langle \tfrac{\mathbf{Y}_i}{\sqrt{D_i}}, \mathbf{u}_i\rangle^2 < \tfrac{(1-\varepsilon)}{(1-\eta)^2}\tfrac{N^2}{r} \mid \neg E_1] \\ &\leq e^{t\frac{(1-\varepsilon)}{(1-\eta)^2}\frac{N^2}{r}} \prod_{i=1}^{N} \mathbb{E}[e^{-t\langle \frac{\mathbf{Y}_i}{\sqrt{D_i}}, \mathbf{u}_i\rangle^2} \mid \neg E_1] \end{aligned}$$

where the first inequality follows from the Markov bound, and we require that $t > 0$. We now follow that analysis in Achiloptas, and expand out the quantity inside the expectation to obtain:

$$\leq e^{t\frac{(1-\varepsilon)}{(1-\eta)^2}\frac{N^2}{r}} \prod_{i=1}^{N} \mathbb{E}[1 - t\langle \frac{\mathbf{Y}_i}{\sqrt{D_i}}, \mathbf{u}_i\rangle^2 + \frac{t^2}{2}\langle \frac{\mathbf{Y}_i}{\sqrt{D_i}}, \mathbf{u}_i\rangle^4 \mid \neg E_1]$$

Now we can apply Achiloptas's results from (52) and (53) to obtain:

$$\leq e^{t\frac{(1-\varepsilon)}{(1-\eta)^2}\frac{N^2}{r}} \prod_{i=1}^{N} \left(1 - \frac{t}{D_i} + \frac{t^2}{2}\frac{3}{D_i^2}\right)$$

and conditioning on $\neg E_1$ gives us:

$$\leq e^{t\frac{(1-\varepsilon)}{(1-\eta)^2}\frac{N^2}{r}} \left(1 - \frac{1}{1+\eta}\frac{tN}{r} + \frac{3}{2(1-\eta)^2}(\frac{tN}{r})^2\right)^N$$

For convience, we'll now let $\tau = \frac{tN}{r}$, and rewrite this as

$$= \left(e^{\frac{(1-\varepsilon)}{(1-\eta)^2}\tau}\left(1 - \frac{1}{1+\eta}\tau + \frac{3}{2(1-\eta)^2}\tau^2\right)\right)^N \doteq \nu(\tau)^N \tag{62}$$

**Bounding equation (62).** We now need to find a choice of $\tau > 0$ that causes (62) to decay with $N$. It will suffice to find $\tau$ that causes $\nu(\tau)$ to decay exponetially, *i.e.,* we want $\nu(\tau) \sim e^{-\chi}$ for some $\chi > 0$. To do this, we start by rewriting $\nu(\tau)$ in the following way:

$$\nu(\tau) = e^{\frac{(1-\varepsilon)}{(1-\eta)^2}\tau}\left(1 - \frac{1}{1+\eta}\tau \cdot \left(1 - \frac{3}{2}\frac{(1+\eta)}{(1-\eta)^2}\cdot\tau\right)\right)$$

Notice that $\nu(\tau)$ is the product of a polynomial and exponential with postive argument (that grows). Notice that the only way we can hope to make $\nu(\tau)$ decay, is if we require the polynomial to decay. To do this, we need to ensure that the expression $(1 - \frac{3}{2}\frac{(1+\eta)}{(1-\eta)^2}\cdot\tau)$ is positive. Thus, we shall choose $\tau = \frac{\varepsilon}{2}(\frac{3}{2}\frac{(1+\eta)}{(1-\eta)^2})^{-1}$. Subsituting in the value for $\tau$ gives us:

$$= e^{\frac{1-\varepsilon}{1+\eta}\frac{\varepsilon}{3}}\left(1 - (\frac{1-\eta}{1+\eta})^2\frac{\varepsilon}{3}\cdot(1 - \frac{\varepsilon}{2})\right)$$

The series expansion of an exponential tell us that for any non-negative $x$ we have the identity $1 - x \leq e^{-x}$. Since the quantity $(\frac{1-\eta}{1+\eta})^2\frac{\varepsilon}{3}\cdot(1-\frac{\varepsilon}{2})$ is non-negative for every $\varepsilon \in (0,1)$, we can apply this identity here:

$$\begin{aligned}
&\leq \exp\left(\frac{1-\varepsilon}{1+\eta}\frac{\varepsilon}{3} - (\frac{1-\eta}{1+\eta})^2\frac{\varepsilon}{3}\cdot(1-\frac{\varepsilon}{2})\right) \\
&= e^{-\frac{\varepsilon}{3(1+\eta)}}\exp\left(\frac{(1-\eta)^2}{1+\eta}(1-\frac{\varepsilon}{2}) - (1-\varepsilon)\right)
\end{aligned} \tag{63}$$

It follows from (63) that proving that $\nu(\tau)$ decays exponentially amounts to ensuring that

$$y(\eta,\varepsilon) \doteq \frac{(1-\eta)^2}{1+\eta}(1-\frac{\varepsilon}{2}) - (1-\varepsilon) \geq 0 \tag{64}$$

and, recalling that $\eta, \varepsilon \in (0,1)$ some MATHEMATICA magic finds that (64) holds as long as $\eta \in (0, c(\varepsilon))$, where

$$c(\varepsilon) = \frac{1}{2-\varepsilon}(3 - 2\varepsilon - \sqrt{5\varepsilon^2 - 14\varepsilon + 9}) \tag{65}$$

This bound on $\eta$, despite being ugly, makes sense. Notice that when $\varepsilon = 0$, we have that $\eta = 0$, and when $\varepsilon = 1$, we have $c(\varepsilon) = 1$ so that $\eta \in (0,1)$. Also, we observe that $y$ monotonically decreases in $\eta$, ranging from $y(0,\varepsilon) = \varepsilon$ to $y((c(\varepsilon),\varepsilon) = 0$.[18] We also observe that $y$ monotonically increase in $\varepsilon$, ranging from $y(\eta,0) = y(0,0) = 0$ (since $\eta = 0$ when $\varepsilon = 0$), and $y(\eta,1) = \frac{1}{2}\frac{(1-\eta)^2}{1+\eta}$ (and $\eta \in (0,1)$ when $\varepsilon = 1$). [19]

Putting everything together, we finally have that as long as $\eta \in (0, c(\varepsilon))$ where $c(\varepsilon)$ is given in (65), then $y$ as given in (64) is such that $y > 0$. Re-writing (62) using (63) and (64) as

$$\Pr[\|\mathbf{w}\|_2^2 < (1-\varepsilon)r \mid \neg E_1] \leq e^{-N\frac{\varepsilon}{3(1+\eta)}y} \tag{66}$$

[18] It's easy to see that when $\eta = 0$, then $y(0,\varepsilon) = \varepsilon$, and a simple check in MATHEMATICA shows that when $\eta = c(\varepsilon)$ as in (65), then $y(c(\varepsilon),\varepsilon) = 0$. By inspection, it's clear that $y$ decreases in $\eta$.

[19] First consider the case where $\varepsilon = 0$. Now when $\varepsilon = 0$, $c(\varepsilon) = 0$, and the requirement that $\eta \in (0, c(\varepsilon))$ implies that $\eta = 0$. It follows that $y = 0$. Next consider the case where $\varepsilon = 1$, which means that for $\eta \in (0,1)$, we have that $y(\eta,0) = \frac{1}{2}\frac{(1-\eta)^2}{1+\eta}$. Now, since the derivative $\frac{dy}{d\varepsilon} = \frac{1+\eta(4-\eta)}{1+\eta} > 0$ for any $\eta \in (0,1)$, we know that $y$ grow monotonically in $\varepsilon$.

we can see that the error decays exponentially in $N$, as required.

### *E.2.1 A simpler statement of the theorem*

We now prove the version of Theorem E.4 that appears in Section 5.4.

THEOREM E.6 (THEOREM 5.2 RESTATED.). *For any vector $\mathbf{v} \in \mathbb{Z}^U$, choose the $N \times U$ matrix $S$ uniformly from $\mathcal{S}_{\mathrm{CCF}}$ and set $\mathbf{w} = S\mathbf{v}$. Then, for all $\varepsilon \in [0,1)$ and $\eta$ such that $\left(\frac{1-\eta}{1+\eta}\right)^2 = \max\left(\frac{1+\frac{\varepsilon}{2}}{1+\varepsilon}\ ,\ \frac{1-\frac{3\varepsilon}{4}}{1-\frac{\varepsilon}{2}}\right)$, choosing*

$$N \geq \tfrac{24}{\varepsilon^2} \ln \tfrac{2}{\delta} \tag{67}$$

$$q, r \geq \tfrac{3N}{\eta^2} \ln \tfrac{4N}{\delta} \tag{68}$$

*ensures that the following two items occur with probability at least $1-\delta$:*

1. *If $\mathbf{v} \in \{-1,0,1\}^U$, and $\|v\|_2^2 \leq q$, then $\|\mathbf{w}\|_2^2 < (1+\varepsilon)q$.*
2. *The number of* non-zero *entries in $\mathbf{v}$ is $r$, then $\|\mathbf{w}\|_2^2 > (1-\varepsilon)r$.*

PROOF. We show how to obtain the Theorem 5.2 from Theorem E.4. To ensure that the error probability is at most $\delta$ in (46) it suffices to set

$$2Ne^{-\frac{\eta^2 q}{3N}} \leq \tfrac{\delta}{2} \tag{69}$$

$$e^{-\frac{N}{2}(y_1^2/2 - y_1^3/3)} \leq \tfrac{\delta}{2} \tag{70}$$

And to ensure that the error probability is at most $\delta$ in (47) we need to set

$$2Ne^{-\frac{\eta^2 r}{3N}} \leq \tfrac{\delta}{2} \tag{71}$$

$$e^{-N\frac{\varepsilon}{3(1+\eta)}y} \leq \tfrac{\delta}{2} \tag{72}$$

**Bounding $N$.** Referring to (70), we need to choose $N > N_{\min,1}$ where:

$$N_{\min,1} = \frac{4}{y_1^2(1 - y_1/6)} \ln \tfrac{2}{\delta} \tag{73}$$

Where recall that $y_1 \doteq \frac{(1+\varepsilon)(1-\eta)}{(1+\eta)^2} - 1$. It's easy to see that $y_1 \in (0, \varepsilon)$ for any $\eta, \varepsilon \in (0,1)$. To simplify (73), we will now require that $y_1 \geq \varepsilon/2$, which means we can write:

$$\begin{aligned} &\leq \frac{4}{y_1^2(1-\varepsilon/6)} \ln \tfrac{2}{\delta} \\ &\leq \frac{4}{(\varepsilon/2)^2(1-\varepsilon/6)} \ln \tfrac{2}{\delta} \\ &\leq \frac{19.2}{\varepsilon^2} \ln \tfrac{2}{\delta} \end{aligned}$$

where the first inequality follows because $y \leq \varepsilon$, the second follows from $y \geq \varepsilon/2$, and the third follows from $\varepsilon \leq 1$. Now, instead of using the "ugly" expression for $N > N_{\min,1}$ in (73) to bound $N$, we have "nicer" bound on $N$ that clearly shows the dependence of $N$ on $\varepsilon, \delta$ as:

$$N \geq \frac{19.2}{\varepsilon^2} \ln \tfrac{2}{\delta} \tag{74}$$

Next, refer to (72), we need to choose $N > N_{\min,2}$ where:

$$N_{\min,2} = \tfrac{3(1+\eta)}{\varepsilon y_2} \ln \tfrac{2}{\delta} \tag{75}$$

Where recall that $y_2 = \frac{(1-\eta)^2}{1+\eta}(1 - \frac{\varepsilon}{2}) - (1-\varepsilon)$. It's easy to see that $y_2 \in (0, \frac{\varepsilon}{2})$ for any $\eta \in (0,1)$. To simplify (73), we will now require that $y_2 \geq \varepsilon/4$ which means we can write:

$$\begin{aligned} &\leq \tfrac{12(1+\eta)}{\varepsilon^2} \ln \tfrac{2}{\delta} \\ &\leq \tfrac{24}{\varepsilon^2} \ln \tfrac{2}{\delta} \end{aligned}$$

where the first inequality follows from our choice of $y_2 \geq \varepsilon/4$ and the second from $\eta \leq 1$. Now we again have "nicer" bound on $N$ (showing it's dependence of $N$ on $\varepsilon, \delta$) as:

$$N \geq \tfrac{24}{\varepsilon^2} \ln \tfrac{2}{\delta} \tag{76}$$

Comparing equations (74) and (76) we find that it suffices to choose $N$ satisfying (76).

**Bounding $\eta$.** These nice bounds on $N$ does not come free. To obtain (74), we need to ensure that $y_1 > \varepsilon/2$. We write

$$\tfrac{\varepsilon}{2} \leq y_1 \doteq \tfrac{(1+\varepsilon)(1-\eta)}{(1+\eta)^2} - 1 \tag{77}$$

$$\tfrac{1+\frac{\varepsilon}{2}}{1+\varepsilon} \leq \tfrac{1-\eta}{(1+\eta)^2} \tag{78}$$

Now since $\frac{1+\frac{\varepsilon}{2}}{1+\varepsilon} \leq \left(\frac{1-\eta}{1+\eta}\right)^2 \leq \frac{1-\eta}{(1+\eta)^2}$ it follows that (78) holds if

$$\tfrac{1+\frac{\varepsilon}{2}}{1+\varepsilon} \leq \left(\tfrac{1-\eta}{1+\eta}\right)^2 \tag{79}$$

Next, to obtain (76) we need ensure that $y_2 > \varepsilon/4$, so we write

$$\tfrac{\varepsilon}{4} \leq y_2 \doteq \tfrac{(1-\eta)^2}{1+\eta}(1-\tfrac{\varepsilon}{2}) - (1-\varepsilon) \tag{80}$$

and a similar argument show that (80) holds as long as

$$\tfrac{1-\frac{3\varepsilon}{4}}{1-\frac{\varepsilon}{2}} \leq \left(\tfrac{1-\eta}{1+\eta}\right)^2 \tag{81}$$

**Bounding $q, r$.** Referring to (69) and (71), we observe that is suffices to choose

$$q, r \geq \tfrac{3N}{\eta^2} \ln \tfrac{4N}{\delta} \tag{82}$$

Notice that this bound relies on both $N$, and $\eta$. We bounded $N$ in (76). To minimize $q, r$, we want to chose $\eta$ as large as possible, subject to the constraints in (79) and (81). Thus, it suffices to chose $\eta$ such that

$$\left(\tfrac{1-\eta}{1+\eta}\right)^2 = \max\left(\tfrac{1+\frac{\varepsilon}{2}}{1+\varepsilon}\ ,\ \tfrac{1-\frac{3\varepsilon}{4}}{1-\frac{\varepsilon}{2}}\right) \tag{83}$$

and this completes our proof Theorem 5.2. $\square$