Towards Measuring Quality of Service in Untrusted Multi-Vendor Service Function Chains: Balancing Security and Resource Consumption*

Prasanna Karthik Vairam†, Gargi Mitra†, Vignesh Manoharan†, Chester Rebeiro†,
Byrav Ramamurthy‡, Kamakoti V†
†*Indian Institute of Technology Madras*, ‡*University of Nebraska – Lincoln*
{pkarthik, gargim, vigneshm, chester}@cse.iitm.ac.in, byrav@cse.unl.edu, kama@cse.iitm.ac.in

***Abstract*—The IT infrastructure of large organizations consists of devices and software services purchased from multiple vendors. The problem of measuring the quality of service (QoS) of each of these vendor devices (and services) is challenging since the vendors may tamper with the measurements for monetary benefits or saving debugging efforts. Existing solutions for QoS measurement in trusted environments cannot be extended for this problem since the vendors can easily circumvent them. Solutions borrowed from other areas such as client-server QoS measurement do not help either since they incur unreasonable storage and network overheads, or require extensive modifications to the packet headers.**

In this paper, we propose the *Measuring Tape* scheme, comprised of (1) a novel data structure called evidence Bloom filter (e-BF) that can be deployed at the vendor devices (and services), and (2) unique querying techniques, which can be used by the administrator to query the e-BF to measure QoS. While e-BF uses storage and computational resources judiciously, the querying techniques ensure resilience to adversarial behavior. We evaluate our solution based on a few real-world and synthetic traces and with different adversaries. Our results highlight the trade-off between resources (i.e., storage and computation) and the accuracy of QoS predictions, as well as its implications on security. We also present an analytical model of e-BF that establishes the relationship between storage, prediction accuracy, and security. Further, we present security arguments to illustrate how our solution thwarts adversarial attempts to tamper QoS.

I. Introduction

Despite the availability of cloud computing platforms, many large corporations and government organizations maintain their own IT infrastructure, for security and privacy reasons. These organizations procure hardware equipment, software services, or a combination of both from several third-party vendors. Although these IT infrastructures have a designated network administrator, such a multi-vendor setup poses many challenges related to management and maintenance of these networked devices and software. Recent innovations in Software Defined Networking (SDN) and Network Functions Virtualization (NFV) have helped alleviate some of these problems using Service Function Chains (SFC). The SFC provides a unified interface from which the connected service elements can be configured on a per-application basis. Figure 1 shows an ordered list of network service elements such as firewall, intrusion detection system, and webserver forming an SFC for web traffic [1].

*A version of this paper has been accepted for publication at IEEE Conference on Computer Communications (INFOCOM) 2019

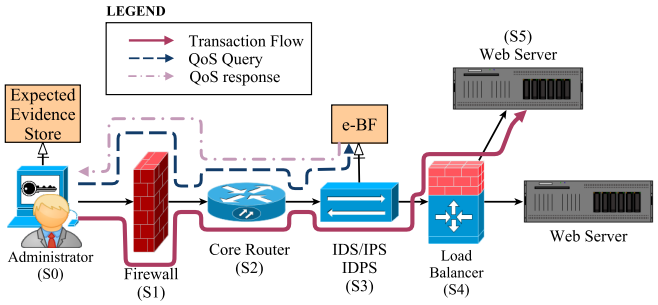


Fig. 1: SFC for a web traffic application is shown. The left most service element is owned by the administrator. The e-BF data-structure is deployed at a service element to evaluate its QoS.

Measuring the Quality of Service (QoS) of each of these service elements has always been a challenge due to the computational and storage resources required. Based on inputs from many banks in our country, we found that there is a dearth of tools to measure the QoS of each vendor reliably in a multi-vendor SFC setup, resulting in vendors blaming each other during debugging. Consequently, this has led to the absence of QoS clauses in Service Level Agreements (SLAs) between the vendors and the organization. However, we argue that the inclusion of QoS clauses in the SLA will improve accountability. For example, web server vendors would guarantee a certain transaction (e.g., HTTP request) processing rate, while firewall vendors may promise an expected accuracy of filtering malicious transactions [2, 3].

Naïvely including QoS clauses in the SLA does not solve this problem, but rather exacerbates it. For instance, to avoid monetary loss [4] due to violations of the QoS clauses, vendors may be tempted to *hide* failures that are either transient or take very less time to fix, thereby reporting an inflated value of their QoS [5, 6]. Vendors typically hide failures by configuring their service element to *lie* that it saw a transaction which, in reality, it did not, or that it did not see a transaction when it actually did. Webservers would benefit from the former, while firewalls could gain from the latter. Identifying SLA violations is an important task. It not only has monetary implications but can also affect the security and reliability of the entire network [4, 5, 6]. Therefore, there is a need for a scheme that can hold the vendors accountable if the QoS clause is violated.

Most prior works [7, 8] for QoS measurement in un-

trusted networks assume that the adversary is located between the sender and receiver. However, when the receiver is also assumed to be untrusted, adapting solutions such as SecureSketch [8] will require large storage at the service elements. Similarly, other existing solutions [9, 10] will require authentication headers to be included in every transaction. This is not a good choice, since **(1)** it consumes payload space that is otherwise available for carrying data; **(2)** it requires payload space proportional to the number of hops in the communication path; and, **(3)** it cannot be easily extended to carry more authentication bits (say, 1024-bits) in case higher security bounds are required.

Contributions: In this work, we propose a novel scheme which can enable the administrator to *reliably* infer if a service element received a transaction, even in untrusted settings, where the service element could be lying. The central idea is that, when queried for the status of a transaction, the service element is required to not only answer that it received the transaction but also provide evidence of it. This is unlike Bloom filter [10, 11] or counter-based techniques [9], wherein the service elements only answer if the transaction is received but do not provide any evidence. To answer such queries, the service element implements a data-structure to store the evidence of transactions. Our solution requires the administrator to be the first element in the SFC so that the administrator can compute the query as well as the evidence it expects from each service element for a particular transaction. This expected evidence can be used to validate the evidence provided by the service element. The proof that a service element *received* the transaction is also the proof that the previous service element in the SFC *forwarded* it.

In principle, hash tables can be used at the service elements to store evidence. For every transaction received, the expected query and the corresponding response (the evidence) for the transaction are stored. However, the storage required for this is considerable, making the scheme impractical. As an alternative, we propose a novel data-structure, called evidence Bloom filter (e-BF), that can be deployed at the service elements to store evidence of transactions more efficiently. The e-BF is an array of cells, where each cell is of a certain number of bits. Unlike hash tables, the query is not stored but instead used as an index to an e-BF cell, which holds the evidence. Each e-BF cell is an accumulator, which can store the evidence corresponding to multiple transactions. This feature permits evidence from a large number of transactions, to be compressed in a small space. For instance, our experiments indicate that a 4.47MB e-BF stores as much information as a 33.125MB hash table.

The use of e-BF as evidence storage gives rise to the following research questions, which we address in this paper.

- Does the space savings in the e-BF result in inaccuracy of stored evidences? Will the QoS evaluation framework be able to handle this error margin?
- Is it possible for an adversarial service element to take advantage of this error margin to forge evidences? If so, how do we deal with it?
- Can an adversarial service element that did not receive a transaction collude with another service element that received it? Similarly, can it deny receiving a transaction?
- What is the accuracy of QoS predictions in adversarial scenarios?
- Is it possible to implement a fool-proof scheme using off-the-shelf hardware-software stacks available today?

The rest of the paper is organized as follows: We discuss the literature related to our work in Section II. We describe the proposed solution and the e-BF in Sections III and IV respectively. The analytical model of e-BF, security arguments, and evaluation are presented in Sections V, VI and VII respectively. Finally, we present the conclusions in Section VIII.

II. Related Work

Earliest works [12, 13, 14] for measuring QoS are designed for generic networks and they assume the network nodes to be honest. These solutions cannot be extended to untrusted settings since an adversary can easily influence the measured QoS [8, 9].

We compare our work with three solutions [8, 9, 10] that can be used in an untrusted setting. SecureSketch [8] implements a data-structure, called *l2-norm*, at the receiver to measure QoS. Although SecureSketch was designed to thwart attempts by adversaries located between a sender and a receiver, it can also be extended to scenarios where the service element, whose QoS is being measured, is dishonest. In such a case, a per-user and per-flow *l2-norm* must be maintained at the service element, requiring a large amount of storage. For example, $150GB$ of storage is required at each SFC node for a 10 Gbps link, as shown in subsequent works [10]. Further, the entire data-structure has to be transported back to the source for quality estimation, resulting in impractical overheads. Subsequently, Faultprints [10] and ShortMAC [9] implement simpler sketching structures such as Bloom filters and counters respectively at the network nodes to reduce the storage requirements. However, both solutions employ authenticator fields in the transaction headers to prevent dishonest nodes from affecting the QoS computed at another node. A major drawback of both schemes is that a dishonest node can still influence its own QoS since it can easily forge its counter values or the 1-bit Bloom filter responses. While ShortMAC assumes that the majority of the network nodes are honest (no forgery and no collusions), Faultprints requires significant modifications to the transaction structure to deal with this problem. Our solution, which uses e-BF, explores the alternative approach of storing more expressive evidences at the network nodes, thereby eliminating the need for authentication fields in the headers that cannot be extended easily to achieve higher security bounds. Also, e-BF has the following advantages when compared to existing solutions [8, 9, 10]: **(1)** it requires lesser storage space at each node, **(2)** it can be shared across flows and users, **(3)** it does not require transportation of the entire e-BF to the administrator for QoS estimation, and **(4)** it is *flexible*, i.e., it can be reconfigured

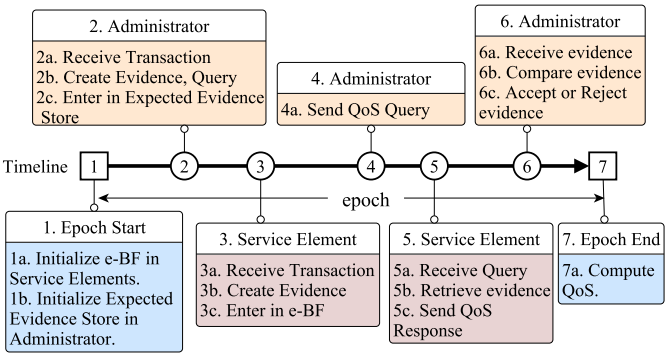


Fig. 2: Operations performed to compute QoS.

every epoch for a particular storage commitment, security bound, and QoS prediction accuracy.

III. Proposed Solution

We consider an SFC of the form $S_1 \rightarrow S_2 \ldots \rightarrow S_r$ where each service element S_i $(1 \leq i \leq r)$ can be owned by a different vendor. One example is shown in Figure 1. A dishonest service element may inflate its QoS to hide SLA violations or faults. The objective of this work is to enable the Administrator to evaluate the SLA compliance of each service element in the SFC. Our solution requires a single trusted service element S_0 at the front of the SFC, denoted as Administrator in Figure 1. The Administrator computes the QoS of service elements based on the feedback (i.e., evidence of transactions) it collects from them.

Each service element in the SFC implements a datastructure, called the evidence Bloom filter (e-BF), which stores the evidence of transactions received. Similarly, the Administrator maintains a data-structure, called the Expected Evidence Store, which stores the QoS query corresponding to each transaction and its expected evidence. Figure 2 shows the timeline of operations performed by the Administrator and a service element. To check if a particular transaction was received by a service element, the Administrator generates the QoS query. The service element responds with the QoS response, which is the evidence of receiving the transaction. This evidence is retrieved from the e-BF maintained by the service element. The Administrator validates the claim of the service element by comparing the QoS response with the expected evidence. The Administrator can only make such queries within a predefined *epoch*, after which the service element may flush the e-BF.

We now describe how the evidence and the query corresponding to a transaction are computed at the i^{th} service element, S_i. Every transaction leaves a *unique* fingerprint at each service element it passes through based on which unshareable evidences are computed, thereby enabling *reliable* QoS measurement. This fingerprint F_i is a function of three factors – **(1)** the constant parts of the transaction, which uniquely identifies a transaction; **(2)** a secret key, $K_{admin,i}$ that is shared between S_i and the Administrator, which uniquely identifies S_i; and, **(3)** the fingerprints $F_{i-1}, F_{i-2}, \ldots F_1$ of the previous service elements in the SFC. These three factors are

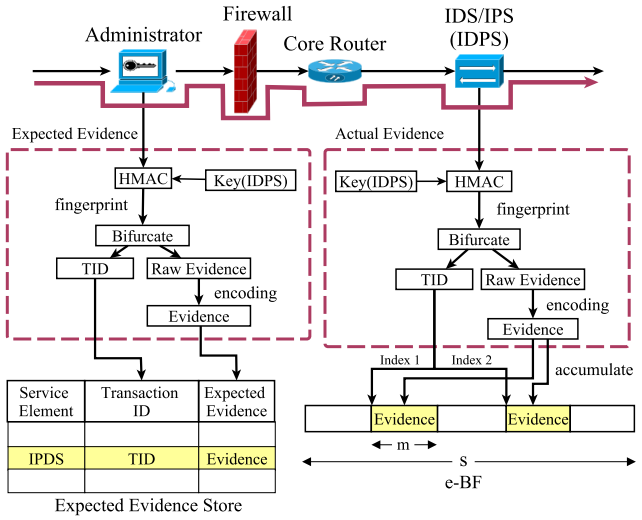


Fig. 3: Proposed solution. Figure shows the operations performed by the Administrator and a service element.

also known to the Administrator, who is therefore capable of computing the same fingerprint, and therefore, the same query and response.

Assumptions. Our solution requires pairwise symmetric keys to be established between the Administrator and each of the service elements of the SFC. Prior theoretical works [15] prove that the secret keys are a necessity for QoS measurement in an untrusted setting and has been used in other works [8, 9, 10]. To avoid missing transactions which arrive at epoch boundaries, we use two e-BF data structures, which are populated and flushed in alternative epochs [10]. The Administrator is assumed to have sufficient storage for the Expected Evidence Store, which could be larger than the e-BF. We also assume that the QoS query and response are sent through multiple paths whenever possible for reliability.

Adversary Model. We assume that the adversary is a service element in the SFC. The adversary is capable of the following: **(1)** guess the evidence for a transaction it did not receive; **(2)** garble the evidence deliberately to disown responsibility for a transaction it received; **(3)** compute evidence for a transaction it did not receive by colluding with another service element in the SFC; and, **(4)** frame another service element for a failure. Attacks such as DoS and DDoS are out-of-scope of this work and there are well-established solutions for handling the same.

A. Measuring Tape: Using e-BF for QoS

In this section, we describe the steps performed for QoS measurement. Figure 3 expands on Figure 1 to highlight the important operations.

Storing evidence in the e-BF at S_i. We first describe how a service element S_i (i.e., IDPS) stores evidence of transactions using the e-BF. The e-BF is an s-bit structure that can be interpreted as n cells (or buckets) each of which is m-bits long and initialized to zeros. The size of each cell (i.e., m) decides the entropy (i.e., resilience to forgery) of the evidence. For every transaction received, the service element S_i performs four steps: **(1)** fingerprint computation; **(2)** fingerprint bifurca-

tion; **(3)** evidence processing and storage; and, **(4)** transaction post-processing.

⇒ *Fingerprint Computation.* When a transaction is received, S_i first computes the 256-bit fingerprint of the transaction using a Hash-based Message Authentication Code (HMAC) function with $K_{admin,i}$ as the key. The HMAC only operates on the constant fields of the transaction and a special field called the *hash chain* field. The value in the hash chain field summarizes the fact that the transaction was forwarded by all the previous service elements in the SFC (i.e., $Admin \rightarrow S_1 \rightarrow \ldots S_{i-1}$) in that order. More details on hash chain computation will be revealed later in this section. The hash chain ensures that the fingerprint of each service element is different, thereby restricting service elements from colluding (Refer Section VI for more details).

⇒ *Fingerprint Bifurcation.* In the next step, the fingerprint is bifurcated into the transaction ID (TID) and the Raw Evidence. The TID is used as an index to the e-BF, while the Raw Evidence is processed and inserted into the indexed cell.

⇒ *Storing in e-BF.* The e-BF is designed so that maximum number of evidences are stored per e-BF cell. The following design choices help achieve this. **(a)** The evidence is inserted into the e-BF cell using an accumulator function, which merges the evidence with the evidences already present in the cell. For example, if bitwise-*or* is used as the accumulator function, every evidence computed is bitwise-*or*'ed with contents of the corresponding e-BF cell; **(b)** we further encode the evidence into a suitable format before storing it;, and **(c)** we overlap e-BF cells so as to further increase the usefulness of each cell. Section IV investigates several accumulator functions, encoding techniques, and e-BF cell overlap methods. Storing multiple evidences in a single cell may affect the accuracy of each evidence stored in it. To improve the accuracy, we store k copies of the evidence in different cells. In such a case, TID would contain multiple indices derived from the fingerprint. Figure 3 shows an e-BF which stores two copies of evidence, i.e., redundancy factor $k = 2$.

At a later point in time, when a service element receives a QoS query, all redundant copies of the evidence will be retrieved from the e-BF using the TID, the most accurate copy will be identified, and sent as a QoS response.

⇒ *Post Processing.* Before forwarding the transaction to the next service element, the value of the hash chain field, denoted by $HashChain_i$, is updated by S_i as follows:

$$HashChain_i = SHA2(HashChain_{i-1}|F_i) \ , \qquad (1)$$

where $SHA2$ is a hash function and '|' is the concatenation operator. The value of $HashChain_0$ is initialized by the Administrator. □

Storing evidence in the Expected Evidence Store. The steps performed by the Administrator for storing evidence is shown in Figure 3. When a transaction is received, the Administrator first fills the $HashChain_0$ field with a random value and computes the exact same fingerprint which the transaction will leave at the service element, S_i. Since the fingerprint at S_i depends on the hash chain value $HashChain_{i-1}$ on the transaction received, it is not straightforward to compute it. The Administrator computes $HashChain_{i-1}$ by recursively computing $HashChain_{i-2}, \ldots HashChain_1$, based on the shared secret keys of $S_{i-2}, \ldots S_1$ (refer Eq 1). The Administrator then follows the first three steps as a regular service element to get the TID and the evidence. The entry stored in the Expected Evidence Store is of the form {service element, TID, expected evidence}. At a later point, the Administrator can use entries from the table to evaluate S_i. □

Validation Function. The QoS response provided by S_i may not match the expected evidence exactly if the e-BF cell accumulated many evidences. The validation function at the Administrator should be able to determine if the response *contains* the expected evidence. Each accumulator function would have its own validation function. For example, if bitwise-*or* is used as an accumulator function, the validation function can check if all the bits set in the expected evidence are also set in the response. If more evidences accumulated in an e-BF cell, it is likely that more number of bits of the response will be set, and the usefulness of the response is reduced. To quantify how close a QoS response is to the expected evidence, we use a metric called *usefulness*, U. For a bitwise-*or* accumulator, usefulness is computed as the percentage of bits in the response (i.e., accumulated evidence) which are not overwritten with 1. This metric represents the confidence of the Administrator on the response provided by S_i. □

QoS Computation. We now describe a novel strategy that reliably calculates the QoS of S_i by automatically neutralizing attempts to $inflate$ QoS. Let us assume that p responses are collected from a service element for computing its QoS. Of these, the ones that fail the validation test are denoted by p_{fail}. The number of evidences which pass the validation test with a high value of usefulness are denoted by p_{pass}. The evidences which pass but have a low value of usefulness, represented by p_{margin}, could arise due to one of the following two cases. It may happen due to many evidences getting accumulated in an e-BF cell or due to QoS inflation attempts by a dishonest service element. For instance, a dishonest service element could forge an evidence with all bits as 1s which will definitely pass the bitwise-*or* validation test but will have zero usefulness.

⇒ *Phantom Queries.* To quantify the extent of QoS inflation, a number of imaginary queries, called *phantom* queries, are sent to the service element. These phantom queries have an imaginary TID and are indistinguishable from regular queries. If a service element is honest, most of its responses corresponding to the imaginary queries will fail the validation test. However, if it is dishonest, it would send forged responses (with many 1s) which will pass the validation test. The QoS $inflation$ of a dishonest service element is measured as the extent to which it *falsely claims* having received such non-existent imaginary transactions.

The aggregate QoS can be calculated after *discounting*

inflation as follows:

$$QoS = \frac{p_{pass}+(1-inflation)p_{margin}}{p_{pass}+p_{margin}+p_{fail}} \ .\square$$

IV. e-BF Design Choices

In this section, we discuss how the parameters and functions of the e-BF must be fixed by the Administrator.

Redundancy factor and accumulator function. The redundancy factor k determines how many copies of evidence must be stored per transaction. A very high value of k may result in the e-BF getting filled faster, while a low value will mean that the probability of finding a least corrupted copy of the evidence is less. Similarly, deciding on the accumulator function, which determines how the existing contents of an e-BF cell are combined with new evidence, is critical. An ideal accumulator should be (i) *inclusive*, i.e., ensure that the evidence already in a cell is never lost, and (ii) *commutative*, i.e., gives the same result irrespective of the order of arrival of transactions. We explored the suitability of functions such as bitwise-and, bitwise-or, bitwise-xor, and chained $hashes$ to serve as an accumulator. We note that bitwise-or is the most suitable accumulator function since it is both inclusive and commutative.

Encoding function. Hamming weight (i.e., the number of 1s in the binary representation) of evidences impacts the usefulness of an e-BF cell. Therefore, we investigate encoding techniques to reduce the Hamming weight, enabling more number of evidences to be stored per e-BF cell. We discuss three encoding functions. **(1)** The *AND* function extracts two m bit strings from the Raw evidence and applies a bitwise-and function to produce the evidence; **(2)** The *COPY* function first extracts two m bit strings from the Raw evidence. Then, it replaces two consecutive 1s from the first with the corresponding bits of the second string to generate the evidence; and, **(3)** NZ-x function which ensures that x is the maximum percentage of 1s in the processed evidence. For instance, NZ-50 has $m/2$ number of 1s in the worst case. Algorithm 1 shows the working of NZ-x. The idea is that NZ-50 chooses $\theta = m/2$ indices of the final evidence that must be set; often resulting in a Hamming weight $< \theta$. The *while* loop sets exactly one bit of the final evidence every iteration (step 5) and runs for θ iterations. Figure 4 compares the encoding functions relative to the Baseline (i.e., no encoding) based on the percentage of good evidences (i.e., Hamming weight $< m/2$), the percentage of unique evidences possible, and the average Hamming weights. The bad evidences have a significant impact on the performance of our scheme, due to which we choose NZ-50 scheme, which always produces good evidences. However, NZ-50 has a significantly less number of unique representations and a slightly higher average Hamming weight than other schemes.

E-BF cell overlap. Adjacent cells of the e-BF can be made to overlap for compressing the evidences further. We denote the number of bits that are shared between adjacent cells of the e-BF by g. Figure 6 shows a few configurations of e-BF by varying g. Contrary to conventional designs [16, 17], which either use $g = 0$ or $g = m - 1$, we observed that $g = m/4$ allows accumulating more evidences per e-BF cell. We formally prove these observations in Section V.

V. Optimal Configuration for the e-BF

In this section, we formally determine the optimal configuration of e-BF to achieve the maximum average *usefulness* U and least *false positive* rate P_{fp}. For this, we first develop an analytical model of the e-BF with cell size m, overlapping bits g, redundancy k, and an encoding scheme with average Hamming weight w. We then present an algorithm in Section V-A that utilizes the analytical model to design the optimal e-BF.

Lemma V.1. *For an e-BF with size s, bucket size m, and bucket overlap g, the number of buckets n is given by,*

$$n = \lceil s/(m-g) \rceil \ . \tag{2}$$

Proof. The number of non-overlapping buckets that fit into a s bit e-BF is s/m. When there is a g bit overlap between buckets, a new bucket starts every $m - g$ bits. ■

Lemma V.2. *The probability $P_{sat,no}$ of a non-overlapping bit in the e-BF turning into 1 (i.e., saturating) after the insertion of p packets is given by,*

$$P_{sat,no} = 1 - \left(1 - \frac{w}{nm}\right)^{pk} \ . \tag{3}$$

Proof. The chance of a bucket being selected for the insertion of an evidence is $1/n$. Also, the chance of a bit of the incoming m-bit evidence being set is w/m. So, the chance of a bit of a bucket getting set to 1 because of the insertion of this evidence is $\frac{1}{n} \times \frac{w}{m}$. Consequently, the chance of a bit not being set after insertion of one evidence is $(1 - \frac{w}{nm})$. Furthermore, the chance of a bit not being set by the insertion of $p \times k$ evidences belonging to p packets is $(1 - \frac{w}{nm})^{pk}$. Therefore, the chance of a bit being set after the insertion of p packets is $1 - (1 - (w/nm))^{pk}$. ■

Lemma V.3. *The probability $P_{sat,o}$ of an overlapping bit in the e-BF turning into 1 (i.e., saturating) after the insertion of p packets is given by,*

$$P_{sat,o} = 1 - \left(1 - \frac{2w}{nm}\right)^{pk} \ . \tag{4}$$

Proof. The chance of an overlapping bit getting set to 1 due to evidences mapping to one bucket is $\frac{1}{n} \times \frac{w}{m}$ (see Lemma V.2). When $1 \leq g \leq m/2$, we can observe that the overlapped bit is affected by evidences mapping to 2 buckets that share it. Therefore, the chances of the bit getting set is $2 \times \frac{w}{nm}$. Rest of the proof is similar to Lemma V.2. ■

Theorem V.4. *The probability P_{sat} of any bit turning into 1 (i.e., saturating) after the insertion of p packets is given by,*

$$P_{sat} = 1 - \left((1 - \tfrac{ng}{s})(1 - \tfrac{w}{nm})^{pk} + (\tfrac{ng}{s})(1 - \tfrac{2w}{nm})^{pk}\right) \ . \tag{5}$$

Proof. From Lemma V.2 and Lemma V.3, we know the P_{sat} for bits that do not overlap and that overlap with other buckets

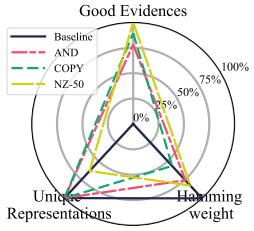


Fig. 4: Comparing encoding schemes

Algorithm 1: NZ-x

input : Raw evidence (RE)
output: evidence (E)

$size \leftarrow \lceil log_2(m) \rceil$;
$i \leftarrow 0$; $it \leftarrow 0$;
while $it < \theta$ **do**
  $ind_{it} \leftarrow int(RE[i : i + size])$;
  $E[ind_{it}] \leftarrow 1$;
  $it \leftarrow it + 1$;
  $i \leftarrow i + size$;
end

Fig. 5: Algorithm for NZ-x Encoding

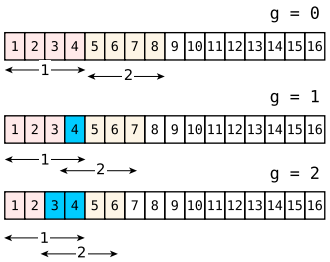
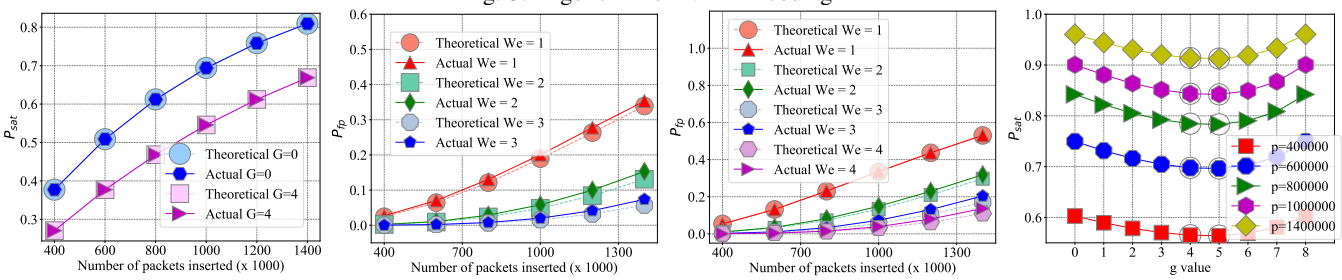


Fig. 6: e-BF layouts: $m = 4$ and $s = 16$.



(a) P_{sat} (b) P_{fp} for NZ30 Encoding. (c) P_{fp} for NZ50 Encoding. (d) P_{sat} by varying g

Fig. 7: Accuracy of the analytical framework. An e-BF with parameters $s = 1024KB$, $m = 8bits$ and $k = 3$ was used.

respectively. Therefore, the P_{sat} for a randomly selected bit is the weighted sum of $P_{sat,no}$ and $P_{sat,o}$. When g bits overlap, ng/s is the fraction of bits in the e-BF that overlap. Consequently, $(1 - ng/s)$ is the number of non-overlapping bits. ■

The accuracy of our theoretical model for P_{sat} is presented in Figure 7a. For various values of p, we measure P_{sat} using simulations and compare it to the value predicted by our analytical framework. We found our framework to be highly accurate with an error rate of less than 10^{-4}.

Theorem V.5. *The expected usefulness $U(\varepsilon)$ of a provided evidence ε, as calculated by the administrator is given by,*

$$U(\varepsilon) \geq (m - w_\varepsilon)(1 - P_{sat}) \ . \tag{6}$$

Proof. The number of non-zero bits in reference ε is $m - w_\varepsilon$. Also, $1 - P_{sat}$ is the probability that a bit of the e-BF remains zero after the insertion of p packets. Therefore, the expected number of zero bits of $\varepsilon_{response}$ that remain zero after the insertion of all the transactions at the service element is $(m - w_\varepsilon)(1 - P_{sat})$. When $k = 1$, this is the expected $usefulness$. When $k > 1$, this becomes a lower bound on the $usefulness$ since the least saturated ε out of the k possible ones is the response. ■

Theorem V.6. *The false positive probability for a query with reference evidence ε is given by,*

$$P_{fp} = \left(\frac{P_{sat}^{w_\varepsilon}}{C_{encoding}}\right)^k \tag{7}$$

Proof. The probability that the bits of a given bucket matches ε is $P_{sat}^{w_\varepsilon}$ if the bits are assumed to be independent. However, since the encoding scheme introduces dependencies among the bits, we add the adjustment constant $C_{encoding}$. Therefore, the P_{fp} i.e., the probability that ε matching with the evidences in all k buckets is $(P_{pfp})^k$, since the choice of k redundant buckets is independent. ■

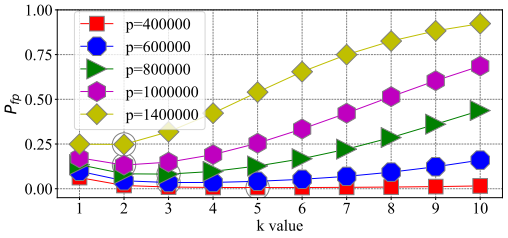


Fig. 8: Effect of k on P_{fp}

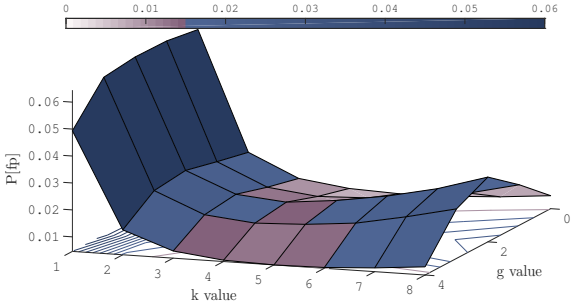


Fig. 9: Effect of k and g on P_{fp}.

The analytical model of P_{fp} is validated against our simulated framework by varying the number of transactions p. Figure 7d and Figure 8 assumes NZ30 and NZ50 encoding scheme which results in queries having a w_ε of $1, 2, 3$ and $1, 2, 3, 4$ respectively. Our analysis is presented separately for each value of w_ε. We found our theoretical model to be consistent with the actual behavior with an error rate of 7%.

Theorem V.7. *The value of P_{sat} achieves a local minima when,*

$$g = m \times (1 - \tfrac{1}{2\ln 2}) \ \textbf{\textit{and}} \ k = \tfrac{mn}{wp} \ln 2 \ . \tag{8}$$

Proof.

$$\frac{\partial}{\partial n}P_{sat} = \frac{\partial}{\partial n}\Big(1 - \Big((1-\frac{ng}{s})(1-\frac{w}{nm})^{pk} + (\frac{ng}{s})(1-\frac{2w}{nm})^{pk}\Big)\Big) \quad (9)$$

Substituting for g from Lemma V.1,

$$= \frac{\partial}{\partial n}\Big(1 - (1-\frac{w}{nm})^{pk} + (\frac{nm}{s}-1)\Big((1-\frac{w}{nm})^{pk} - (1-\frac{2w}{nm})^{pk}\Big)\Big)$$
$$\approx \frac{\partial}{\partial n}\Big(1 - e^{\frac{-wpk}{nm}} + (\frac{nm}{s}-1)(e^{\frac{-wpk}{nm}} - e^{\frac{-2wpk}{nm}})\Big)$$
$$= \frac{-wpk}{mn^2}e^{\frac{-wpk}{nm}} + (\frac{nm}{s}-1)(-\frac{wpk}{mn^2}e^{\frac{-wpk}{nm}} + \frac{2wpk}{mn^2}e^{\frac{-2wpk}{nm}}) + \frac{m}{s}(e^{\frac{-wpk}{nm}} - e^{\frac{-2wpk}{nm}})$$

The above differential is 0 when we substitute $\frac{kpw}{nm} = \ln 2$ and $n = \frac{2s}{m}\ln 2$. Also, $\frac{\partial^2}{\partial^2 n}P_{sat} > 0$ indicating that this point is a minima. Again, substituting for n from Lemma V.1, we get $\hat{g} = m \times (1 - \frac{1}{2\ln 2})$ and $\hat{k} = \frac{mn}{wp}\ln 2$. ■

Figure 7d shows that P_{sat} reaches a minima between $g = 4$ and $g = 5$ independent of the number of packets inserted. This is consistent with the prediction of $g = 4.46$ from our theoretical model from Theorem V.7. Note that $U(\varepsilon)$ also reaches its maxima at this point according to Theorem V.5.

Theorem V.8. *The false positive rate P_{fp} achieves a local minima when,*

$$k = \frac{mn}{wp}\ln 2 \quad \textbf{\textit{and}} \quad g = 0 \textit{(or)} m-1 \ . \quad (10)$$

Proof.

$$\frac{\partial}{\partial k}\ln P_{fp} \approx \frac{\partial}{\partial k}\ln P_{sat}^{k*w_\varepsilon} = \frac{\partial}{\partial k} w_\varepsilon * k \ln P_{sat}$$
$$= w_\varepsilon\Big(\ln P_{sat} + \frac{k}{P_{sat}}\frac{\partial}{\partial k}P_{sat}\Big)$$
$$= w_\varepsilon * P_{sat} * \ln P_{sat} + w_\varepsilon * k * \frac{\partial}{\partial k}P_{sat}$$

To find the minima, $\frac{\partial}{\partial k}\ln P_{fp} = 0$,

$$= P_{sat} * \ln P_{sat} + k * \frac{\partial}{\partial k}P_{sat} = 0$$

When $\frac{kpw}{mn} = \ln 2$,

$$= \frac{1}{4}\Big(\frac{mn}{s}+1\Big) * \ln\Big(\frac{mn}{s}+1\Big) + \frac{1}{2}\ln 2 = 0$$

We can see that $\frac{\partial}{\partial k}P_{fp} = 0$ when $\frac{kpw}{mn} = \ln 2$ and $mn = s$. Also, $\frac{\partial^2}{\partial^2 n}P_{fp} > 0$ indicates that it is a minima. Therefore, the P_{fp} is minimized when $\hat{g} = 0$ (or) $m-1$ (which gives $mn = s$) and $\hat{k} = \frac{mn}{pw}\ln 2$. ■

Figure 8 shows that P_{fp} reaches its minima at $k = 5, 3, 3, 2, 1$ for increasing values of p. Our theoretical model presented in Theorem V.8 predicts $k = 4.39, 2.92, 2.19, 1.75, 1.25$ for the respective values of p.

Figure 9 shows that P_{sat} and P_{fp} are minimized for different values of k and g. By allowing overlaps between the buckets (when $g = m \times (1 - 1/(2\ln 2)) \approx m/4$), the false positive rate is slightly away from the minima while the usefulness reaches its maxima. This is because, the overlapped bits have a higher concentration of 1s and increase the false positive rate of queries. At the same time, the non-overlapped bits of the bucket have a lesser concentration of 1s thereby improving the usefulness. Therefore, we conclude that the e-BF designer has to choose between optimizing P_{fp} and $U(\varepsilon)$.

Algorithm 2: Optimal e-BF configuration

input : Storage (s), Usefulness threshold (U_{th})
input : False positive threshold (FP_{th}), Encoding HW. (w)
output: e-BF parameters m, g, k

```
p ← Number of transactions in previous epoch;
m ← arbitrarily high number; U ← 0; P_fp ← 1; g = m/4;
while U < U_th and P_fp > FP_th do
    n ← ⌈s/(m − g)⌉  k ← mn ln 2/wp;
    Recalculate P_fp and U;
    m ← m/2;
end
m ← m × 2;
return m, g, k
```

Fig. 10: Algorithm for Optimal e-BF configuration

A. Algorithm for Applying the Analytical model

The Algorithm 2 shows the steps to configure the e-BF every epoch. It works to meet a given storage budget (s), usefulness threshold (U_{th}), and a false positive threshold (FP_{th}). The encoding scheme is fixed beforehand and assumed to have a Hamming weight of w. The level of security i.e., m is set to a maximum value and exponentially reduces until the U_{th} and FP_{th} are met. Every iteration, g, k, and n are set to optimal values based on the analytical model.

VI. Security Arguments

In this section, we discuss potential threats from a dishonest service element S_i and how they are thwarted.

Guessing the evidence for a transaction. A dishonest service element S_i may try to guess the evidence for a transaction which it did not receive. Forging an evidence with higher Hamming weight will increase the chance of passing the validation function but it will result in low usefulness; and vice-versa. We handle both these cases in our scheme. The extent of QoS inflation is quantified using phantom queries (refer Section III-A) and discounted during QoS computation.

Security of phantom queries. A dishonest service element cannot distinguish between phantom queries and regular queries since they look the same. The use of Bloom filters for storing received transactions will not help either since both phantom transactions and lost transactions will not have an entry in the Bloom filter.

Colluding Service elements. A dishonest service element S_i may falsely claim to have received a transaction by colluding with another service element (say S_j, where $j < i$). The service element S_j will attempt to compute the same fingerprint as S_i would. Though the secret key of S_i and the constant fields of the transaction are available to S_j, computing the value of $HashChain_i$ is difficult since it depends on $HashChain_{i-1}$. This cannot be computed unless the secret keys of S_{j+1} to S_{i-1} in the SFC are known to S_j. This is *impossible* if there exists at least one non-colluding service element between the two. Therefore, we can limit the repercussions of collusions, which is a significant improvement over the state-of-the-art [9].

Receive and drop attacks. The proof of receipt given by an S_j, where $j > i$ is the proof that S_i actually forwarded it. If

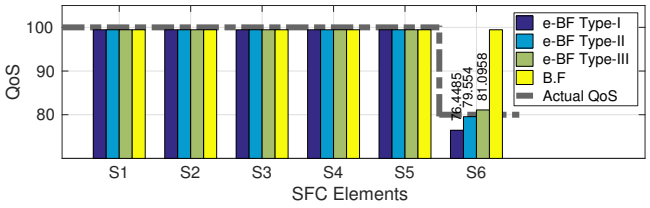


Fig. 11: A comparison of the accuracy of the QoS predictions in various adversarial conditions is shown. The SFC considered is from Figure 1.

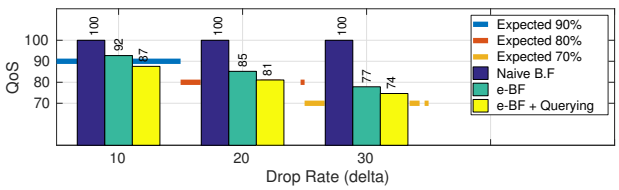


Fig. 12: Effectiveness of components of our solution.

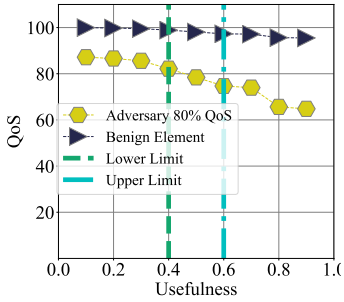


Fig. 13: Fixing the right *usefulness* threshold

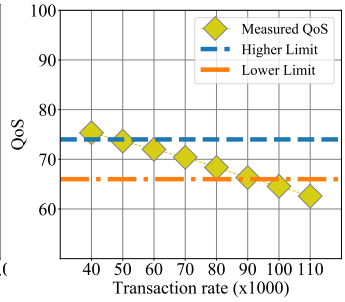


Fig. 14: Transaction rates supported by an e-BF

Storage →	$s = X$		$s = 2X$		$s = 5X$	
Security ↓	Benign	Adv.	Benign	Adv.	Benign	Adv.
$m = 4$	81.5%	91.6%	81.1%	89.2%	80.7%	85%
$m = 8$	79.1%	85.1%	80.8%	84.4%	80.7%	83.4%
$m = 16$	74.7%	72.3%	**80.3%**	**78.9%**	80.7%	80.1%

TABLE I: Trade-off between Security (m) and Storage (s). Ground truth QoS is 80%. Measured QoS values are shown.

S_i is dishonest, it will only be able to transfer the failure from the link $S_{i-1} \rightarrow S_i$ to the link $S_i \rightarrow S_{i+1}$. This is in line with prior theoretical works [15].

Secret Parameters and effect of leaks. The e-BF parameters as well as the shared secret key with the Administrator to be changed at epoch boundaries for this.

VII. Evaluation and Discussion

In this section, we discuss the evaluation framework and a case study on measuring QoS of a web server. The e-BF data-structure and the associated schemes are developed as Python modules. We used multiple topologies and anonymized real-world *https* traces collected from a website for evaluation. We also cover other scenarios (e.g., variation in transaction size and rate) using synthetic traces. The ground truth QoS value is used as a reference. The objective is to analyze the ability of the proposed solution to match the ground truth QoS values even when the service element lies.

An e-BF of size $4.47MB$ with cell size $m = 8$ bits, redundancy factor $k = 2$, cell overlap $g = 2$ is used. NZ-50 encoding is assumed for the evidence. Such an e-BF is capable of providing usefulness of at least 60% and an error rate $< 3\%$. A Bloom filter of size 3.65MB with a *false positive* rate $< 3\%$ is used for comparison with the e-BF. We evaluate our scheme at a transaction rate of $50,000$ requests per second, which is comparable to workload intensive real-world setups such as Wikipedia.

A. Case Study: Web server

We consider the SFC for web applications such as the one shown in Figure 1. The webserver S_6 is assumed to have a flaw, which results in $\delta\%$ of transactions getting dropped. We assume that the service elements in the SFC could be dishonest and implement an e-BF. For comparison, we also consider the scenario where the service elements implement a Bloom filter.

Modeling the adversary. We consider three types of adversarial service elements. Type-I forges evidences such that its probability of passing the validation test is maximized. For instance, it could forge evidences which have all bits set to 1, i.e., Hamming weight of 8 for $m = 8$. Type-II and Type-III forge evidences which have more usefulness compared to Type-I but may not always pass the validation test. Type-II and Type-III do so by forging evidences with a Hamming weight of 7 and 6 respectively.

Results. Figure 11 shows the QoS of the service elements in the SFC for different adversaries and $\delta = 20\%$ at S_6. For lost transactions, the Bloom filter case simply replies with "transaction received", thereby successfully hiding the failure. Thus, as seen in Figure 11, it can falsely project 100% QoS despite failures. However, when e-BF is used at $S6$, the QoS measured is $\approx 80\%$. The maximum inflation achieved is only 1% (by the Type-III adversary).

Further, Figure 12 shows the extent to which the e-BF structure and phantom queries contribute to the QoS prediction accuracy in the case of Type-III adversary. For $\delta = 20$, with just e-BF, the maximum inflation in QoS is 5%. Phantom and positive queries help narrow it down to 1%.

The threshold value of *usefulness* is fixed at 60%. A high threshold will underestimate the QoS of benign service elements, while a lower threshold will make it easy to cheat. A rough guide to fixing the threshold value (i.e., $0.4 <= threshold <= 0.6$) is presented in Figure 13. We also evaluate the resilience of the e-BF to a sudden increase in the transaction arrival rate. Figure 14 shows that the QoS prediction accuracy is within acceptable limits for up to $90,000$ requests per second on an e-BF designed to handle $50,000$ requests per second. The transaction sizes and topology do not affect the results, as expected.

Table I shows the measured QoS (both in benign and adversarial scenarios when ground truth QoS is 80%) for different combinations of storage space s and level of security m. To establish a baseline for the maximum achievable QoS prediction accuracy, we consider $s = 5X$, where X is the storage budget. In this case, we can observe that $m = 16$

provides the best QoS in adversarial conditions as expected. When $s = X$, the following observations can be made: (1) Both $m = 4$ and $m = 8$ do not provide enough security, resulting in inflated QoS in adversarial scenarios; (2) $m = 16$ ends up saturating the e-BF resulting in conservative QoS values (¡ 80%). When the storage is doubled to $s = 2X$, setting $m = 16$ gives the most accurate QoS estimate.

B. Overheads.

The e-BF must store $P = 50,000$ requests/second $\times$ 100 $seconds$ number of transactions, which takes affordable in-memory storage of about $4.47MB$ per service element to maintain a prediction accuracy of 97%. On the other hand, a comparable hash table requires 53 bits $\times$ $50,000$ requests/second $\times$ 100 seconds, where the combined size of TID and evidence is 53 bits; resulting in $33.125MB$ of storage.

Our experiments indicate that over 6.5 million OpenSSL [18] SHA-256 hash digest computations can be performed per second for 1024 Byte transactions on a multi-core Intel Core-i7 processor. This is sufficient for both the Administrator and the service elements (using one core).

C. Limitations and Future work

Our solution restricts collusion between service elements to a great extent but cannot eliminate it. While it is possible to solve this by dynamically interleaving a few trusted service elements in the SFC chain, the overheads incurred will be high. We believe that Physically Unclonable Functions (PUFs [19]) can be used instead. The PUF is capable of tying a device to an *unclonable evidence*, thereby eliminating collusions.

The proposed solution also opens up the following research directions, some of which are both important and interesting.
Stochastic nature of e-BF. Since the arrival of transactions and queries into e-BF is non-deterministic, an analysis of the stochastic nature of e-BF is required. It may, for instance, help answer questions like 'Is it advantageous for an adversary to forge evidences at the starting or towards the end of the epoch?'.
Hardware. An interesting direction is to see if running the e-BF in an encrypted enclave such as Intel SGX [20] can add value in terms of security. Other enhancements include accelerating the e-BF primitives using an FPGA and using the Physically Unclonable Function (PUF [19]) response in evidence calculations.
Networking. We have demonstrated the applicability of e-BF using a simple SLA use-case. Implementing more complex SLAs such as those involving delay guarantees could be worth exploring.

VIII. Conclusions

This paper addresses the problem of reliably measuring QoS (with 97% prediction accuracy) in a multi-vendor SFC setting using a novel data-structure, called e-BF, that uses storage resources judiciously. We note that better QoS prediction accuracy can be achieved by allocating more storage. Prior works have explored the storage space vs. QoS prediction accuracy trade-offs (using Bloom filters). To the best of our knowledge, this is the first work to analyze the trade-off between storage space, security, and QoS prediction accuracy. We believe that the proposed solution is the first step towards solving this problem and reach out to the research community to enhance the security guarantees.

Acknowledgement

This work is supported by DST-FIST Grant Program 2016, from Department of Science and Technology, India. Professor Ramamurthy was supported by NSF Grants (NSF OAC-1541442 and CNS-1817105). Thanks to Kris Gopalakrishnan Endowment and Microsoft Research for their travel grants.

References

[1] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015.

[2] "IBM Managed Security Services for Network Firewalls," accessed: 2018-01-5. [Online]. Available: https://www-935.ibm.com/services/us/igs/pdf-iss-contracts/ireland-7799-00.pdf

[3] "Dedicated Service Level Agreement," https://www.liquidweb.com/about-us/policies/dedicated-sla/, accessed: 2018-01-5.

[4] "Service Level Agreements Understanding Practical Remedies in Data Center Leases," accessed: 2018-01-5. [Online]. Available: http://www.datacenterknowledge.com/industry-perspectives/service-level-agreements-understanding-practical-remedies-data-center

[5] "Summary of the October 22, 2012 AWS Service Event in the US-East Region," accessed: 2018-01-5. [Online]. Available: https://aws.amazon.com/message/680342/

[6] "Summary of Windows Azure Service Disruption on Feb 29th, 2012," accessed: 2018-01-5. [Online]. Available: https://azure.microsoft.com/en-us/blog/summary-of-windows-azure-service-disruption-on-feb-29th-2012/

[7] J. Sommers, P. Barford, N. Duffield, and R. Amos, "Accurate and Efficient SLA Compliance Monitoring," *SIGCOMM Comput. Commun. Rev. (2007)*.

[8] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford, "Path-quality monitoring in the presence of adversaries," ser. SIGMETRICS '08. New York, NY, USA: ACM, 2008, pp. 193–204.

[9] X. Zhang, Z. Zhou, H.-C. Hsiao, A. Perrig, and P. Tague, "ShortMAC: Efficient data plane fault localization," in *NDSS*, 2012.

[10] C. Basescu, Y.-H. Lin, H. Zhang, and A. Perrig, "High-speed inter-domain fault localization," in *IEEE S&P (2016)*.

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] J. Sommers, P. Barford, N. Duffield, and A. Ron, "Improving accuracy in end-to-end packet loss measurement," ser. SIGCOMM '05. New York, NY, USA: ACM, 2005, pp. 157–168.

[13] E. Stephan, "IP Performance Metrics IPPM Metrics Registry," Internet Requests for Comments, RFC Editor, BCP 108, August 2005.

[14] B. Claise, A. Johnson, and J. Quittek, "Packet sampling PSAMP protocol specifications," Internet Requests for Comments, RFC Editor, RFC 5476, March 2009.

[15] B. Barak, S. Goldberg, and D. Xiao, "Protocols and lower bounds for failure localization in the internet," in *EUROCRYPT (2008)*.

[16] H. Dai, Y. Zhong, A. X. Liu, W. Wang, and M. Li, "Noisy bloom filters for multi-set membership testing," ser. SIGMETRICS '16. New York, NY, USA: ACM, 2016, pp. 139–151.

[17] S. Xiong, Y. Yao, Q. Cao, and T. He, "kBF: a bloom filter for key-value storage with an application on approximate state machines," in *IEEE INFOCOM 2014*, April 2014, pp. 1150–1158.

[18] "OpenSSL Speed Test," accessed: 2018-04-1. [Online]. Available: https://www.openssl.org/docs/manmaster/man1/speed.html

[19] C. Herder, M. D. Yu, F. Koushanfar, and S. Devadas, "Physical unclonable functions and applications: A tutorial," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1126–1141, Aug 2014.

[20] "Intel Software Guard Extensions (Intel SGX)," https://software.intel.com/en-us/sgx, accessed: 2018-04-1.