Bounding Packet Dropping and Injection Attacks in Sensor Networks

Xin Zhang, Haowen Chan, Abhishek Jain and Adrian Perrig

November 9, 2007
CMU-CyLab-07-019

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Bounding Packet Dropping and Injection Attacks in Sensor Networks

Xin Zhang, Haowen Chan, Abhishek Jain and Adrian Perrig
Carnegie Mellon University

Abstract

A malicious insider in a sensor network may sabotage the network at any level of operation. While most prior work on network-layer security has focused on providing control plane integrity (specifically, routing correctness), we approach a complementary and equally important problem: data plane reliability. In a data plane attack, the attacker does not attack the routing control logic, but instead directly manipulates the data payloads flowing on the network to cause disruption. We reduce the general problem of data-plane manipulation attacks to two specific attacks: packet dropping and injection, and propose two complementary protocols to address the problem as a whole. We address packet dropping with a probabilistic probing protocol which can bound the end-to-end drop rate below a fixed threshold for a given path in the presence of multiple adversarial nodes and natural packet loss. We address packet injection with a rate-limiting mechanism based on per-epoch audit to detect nodes which exceed their allotted data origination rates. In both protocols, an adversary can misbehave by at most a fixed amount on expectation, before it is detected; after detection, one of the links under its control will be removed. Hence, the total amount of misbehavior (packet injection or dropping) an adversary can inflict is a constant regardless of the lifetime of the network.

1 Introduction

In sensor networks, attacks on the network layer can be divided into *control plane* attacks, which aim to disrupt the routing logic of the network; and *data plane* attacks, which leave the routing structure unchanged but instead manipulate packet flows. Data plane attacks include packet dropping, packet injection and packet alteration. Data plane security represents a complementary and equally important security property for networking reliability in sensor networks: no secure routing protocol can defend against data plane attacks, and routing correctness is less useful if the data itself can be arbitrarily sabotaged by the adversary.

In this paper we propose a comprehensive set of algorithms for bounding the adversary's ability to perform data plane attacks in sensor networks. The goal is to provide a *provable, guaranteed* level of useful data throughput even in the presence of multiple adversarial nodes within the network.

Given a correctly functioning communications infrastructure, data plane attacks include anything an adversary could do at the OSI network layer without attacking control aspects such as address lookup or routing. Specifically, an adversarial node within the network can misbehave in two main ways: packet dropping and injection. Other attacks include packet alteration (which, if we assume that data is end-to-end authenticated, is equivalent to a packet drop followed by a packet injection); packet replay (reduced to packet injection if sequence numbers are implemented); and packet delay (which is typically an attack on a specific protocol and hence it is impractical to devise general countermeasures; in some scenarios, long packet delay may be treated as packet drop).

Based on these observations, we identify packet dropping and injection as the two basic threats to data-plane security in sensor networks. Packet dropping directly reduces the amount of legitimate throughput; packet injection may reduce network functionality by sapping battery power and causing premature node death, or by causing sufficient network congestion to cause packets to be dropped due to resource contention. Security against packet dropping and injection are mutually dependent; specifically, when facing a rate-limiting mechanism without a packet-dropping detection mechanism, an adversary could cause resources to be wastefully consumed without changing its own rate of packet origination, by altering packets randomly and forwarding these useless packets to the base station and forcing wasteful retransmissions from the source. On the other hand, if a packet-dropping detection mechanism is implemented without a rate-limiting mechanism, then an adversary could cause legitimate links to be detected as faulty by flooding these links with spurious messages to deny them the battery or memory resources necessary to cope with the legitimate traffic. Our goal is to provide a suite of algorithms where, provided that routing functions correctly, then some level of useful throughput of the network can be assured. For this to

be achieved, we need to simultaneously bound the adversary's ability to perform either packet injection or packet dropping.

While there has been substantial prior work on various aspects of detecting malicious links in a network, there currently exists no scheme which is fully practical in the sense of being both accurate in the presence of high natural packet drop rates, and being optimised for efficiency in the case where the network is not under attack: to address this, we present the first malicious packet-dropping detection protocol to achieve both these goals. This protocol is based on the crucial observation that existing methods of link-probing require a large number of simultaneous audits of various nodes along the path to pin down the location of the packet loss; by performing only a single random probe per packet loss we are able to achieve low overhead while preserving high accuracy. The complementary problem of packet injection has only been studied in the context of control-plane attacks; we present the first packet injection defense aimed at limiting the rate of data injection; we do this by reducing the rate-limiting problem to the secure-summation problem: the key observation is that in both problems, we have internal nodes in the network claiming some input (i.e. the amount of incoming data flows) in order to justify some output value (i.e. the amount of outgoing data flow). We can then adapt a well-known secure-summation algorithm to solve the data injection problem. When executed together, the combined protocols preventing packet dropping and injection represent the first comprehensive protocol which quantitatively bounds the amount of influence the attacker can have on the data in the network. Our contributions are summarized as follows.

Our Contribution. (a) We propose that a complete data-plane security protocol for sensor networks must address both packet dropping and injection attacks, and present a protocol suit to address both simultaneously. (b) To the best of our knowledge, our protocol is the first to provide a provable, guaranteed level of useful data throughput in the presence of strong adversary within the sensor network: we first bound the adversary's ability to drop packets under a fixed threshold before being detected; we then ensure that the adversary can inject at most a fixed volume of data beyond its allotted quota; and all together we bound the total amount of traffic that the adversary can disrupt. (c) We propose the *first* protocol to localize packet dropping in sensor networks with rigorous theoretical analysis in the presence of natural packet loss. (d) Next, we present the *first* protocol for sensor networks to bound the rate of packet injection by malicious nodes.

Organization. We begin by stating our general assumptions in Section 2. Next, we describe the construction of our PAI protocol in Section 3, following which we provide a theoretical analysis of PAI in Section 4. We present our protocol for preventing packet injection attacks in Section 5 and 6. We present the combined analysis of both our protocols in Section 7. We discuss the related works in Section 8. Finally, we summarize our work and conclude in Section 9.

2 Assumptions

2.1 Network Assumptions

We assume a general multi-hop network with a set $\phi = \{f_1, \ldots, f_n\}$ of n sensor nodes and a trusted base station. We assume a directed acyclic graph (DAG) topology directed towards a single sink (the base station) which we call the *root* of the DAG. All messages are assumed to be destined for the base station – for simplicity we ignore factors such as data aggregation and assume that each message is forwarded without modification to the base station. We call nodes closer to the base station *downstream* nodes, and nodes that are further away from the base station *upstream* nodes. Since the DAG topology is directed towards the root, for any node x we can define the set of *ancestors* of x to be the set of all nodes downstream from x (i.e., reachable via a path from x to the base station) and similarly a set of *descendants* of x which are the set of nodes for which x is an ancestor. For a given source and destination pair, we assume the presence of *symmetric paths*, where the forward path (for data) and reverse paths (for acknowledgements) are identical. Finally, we assume that the nodes are loosely time-synchronized.

2.2 Security Infrastructure

Given a path from a source to the base station, we assume that the source shares a unique symmetric key with each intermediate node on the path to the base station. Any known key establishment protocol may be used to establish pairwise keys; for example elliptic curve PKI-based methods could be used at initial deployment-time [11]. We further assume the existence of a broadcast authentication primitive where any node can authenticate a message from the base station. This broadcast authentication could, for example, be performed using μTESLA [16]. We assume the sensor nodes have the ability to perform symmetric key operations as well as computations of a collision

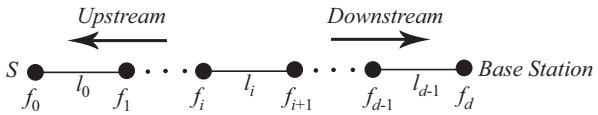


Figure 1: Example topology

resistant hash function h and a keyed pseudorandom function PRF.

2.3 Adversary Model

We assume an adversary with polynomially bounded computation who is in complete control of an *arbitrary number* of sensor nodes, including knowledge of their secret keys. The adversary can eavesdrop and perform traffic analysis anywhere in the network. We assume hop-by-hop authentication so that packet injection at an arbitrary link is not generally feasible. However, the adversary may intercept and inject messages at will on the links that are under its control. The goal of the adversary is to degrade the network throughput by dropping or injecting packets while avoiding detection.

3 Bounding Packet Dropping Attacks

In this section, we first give our problem definition and the metrics we use to evaluate the performance of the protocols. Next, we present a strawman approach to motivate the construction of our PAI protocol. Finally, we present our PAI protocol in Section 3.2, followed by a rigorous theoretical analysis in the next Section.

Problem Definition. Given a set of adversarial nodes located on the forwarding path from a source node to the base station, we are interested in the design of protocols that allow the source to monitor the forwarding path for packet dropping activity over a period of time and then securely decide whether a particular link is malicious. We build on the well-known approach of requiring acknowledgement packets (ack) from the intermediate nodes on the forwarding path. We assume that links on the forwarding path exhibit some natural packet loss. Further, the adversarial nodes on the path may collude, and try to bias the measurement results (from the knowledge of the protocol parameters) in order to evade detection or incriminate honest links as malicious.

Problem Metrics. In our discussion, we do not consider protocols that employ asymmetric key cryptography. Therefore, assuming that the protocols utilize symmetric key cryptography with a reasonable computational overhead, we identify two key metrics to evaluate the protocols, (a) communication overhead due to ack-size, and (b) detection rate, which is defined as the number of packet transmissions required to detect a malicious link with the false positive and negative below a certain threshold.

Notation. We focus on one source S and a given path of length d hops to the base station. The nodes along the path are denoted as $f_0, \ldots, f_d$, where f_i is i hops away from S (see Figure 1). Let l_i be the link between node pair f_i, f_{i+1}. S shares a pairwise symmetric key K_i with each intermediate node f_i. Let $\mathrm{E}_K(\cdot)$ denote encryption and $\mathrm{MAC}_K(\cdot)$ denote MAC computation with symmetric key K. For simplicity, in our description, we do not differentiate between the keys for encryption and MAC computation; although in practice, one would derive separate keys for encryption and MAC computation from a common key. We denote the round-trip time from a node f_i to the base station as r_i.

3.1 A Strawman Approach: Full-ack Scheme

Assuming that the links on the forwarding path exhibit some natural packet loss, we first explain the full-ack protocol that is capable of identifying a malicious link over a period of time even if multiple (colluding) adversarial nodes exist on the path. For any dropped packet, the full-ack scheme can determine precisely the location of the drop; thus it is able to directly compute the drop rate of each link on a given path and identify malicious links within a small number of packet drops. However, the high detection rate is achieved at the price of a large communication overhead ($\mathrm{O}(d)$ ack-size) per data-packet sent out by the source, *irrespective of whether the packet was dropped on the forwarding path*. In contrast, our PAI protocol (explained later) employs a probabilistic sampling approach to gather only approximate evidence about the location of the drop. Therefore, it requires more drops to occur before adequate evidence can be accumulated over a period of time to make an accurate decision on whether a link is malicious; however, in the process, we reduce the communication complexity to $\mathrm{O}(1)$ per data packet sent out

by the source.

We now describe the full-ack scheme in detail. Consider the example topology in Figure 1. Let $h(m)$ be a packet identifier for any data packet m. We define an ack packet with the structure $a_i = \langle h(m)||\mathcal{A}_i\rangle$, where, $h(m)$ serves as an identifier for data packet m and $\mathcal{A}_i$ is a report computed by node f_i for that data packet.

Now, let us consider that S sends out a data packet m towards the base station. On receiving a packet m, an intermediate node f_i starts a wait-timer and forwards m to its downstream neighbor. The base station must return an ack to S on receiving any data packet. Specifically, it computes a report $\mathcal{A}_d$ as a MAC over the identifier $h(m)$ using the secret key K_d shared with S, and sends out an ack $a_d = \langle h(m)||\text{MAC}_{K_d}(h(m))\rangle$. On receiving an ack a_{i+1} from its downstream neighbor, an intermediate node f_i extracts the report A_{i+1} and then computes its own report as a MAC over the concatenation of the identifier $h(m)$ and the downstream report A_{i+1} using the secret key K_i shared with S. It then sends out an ack $a_i = \langle h(m)||a_{i+1}||\text{MAC}_{K_i}(h(m)||a_{i+1})\rangle$ towards S. However, if no ack is received within the wait-time, f_i creates a fresh ack in a manner similar to the base station and sends it towards S. Now, on receiving the final ack, S can sequentially verify each report embedded in it. For some $i < d$, if the MAC from each intermediate node $f_j, j \in [1, i]$ is valid but the MAC from f_{i+1} is invalid or not present in the final ack, then S concludes that either the data packet m, or an ack was dropped (or altered) at the link l_i. It therefore adds one to the *drop score* of l_i. The actual reliability of a link can be derived from its drop score via a simple computation against the drop scores of the other links in the path: intuitively, links with higher drop scores are considered more unreliable.

Suppose multiple adversarial nodes are present on the path who share their keys. Let f_i be the malicious node located closest to S on the path. If f_i drops or alters a data packet, no valid report will be created by any honest node downstream to f_i. Since f_i cannot forge MACs for honest nodes, it must either (a) create a fresh ack or use the invalid report from downstream ack to create its own ack, or (b) do not ack. In the first case, S will determine that link l_i is faulty and add one to its drop score, while in the second case node f_{i-1} will return a valid ack (since f_{i-1} is honest) and therefore S will determine link l_{i-1} as faulty and add one to its drop score. Note that the knowledge of the keys of other adversarial nodes does not help f_i since it cannot forge MACs for honest nodes located between f_i and any other adversarial node. Further, observe that if a malicious node f_i drops or alters a downstream ack, it will be detected by S during the final ack verification.

In summary, if a malicious node drops or alters a packet (data or ack), one of its adjacent downstream links has its drop score increased. Over a period of time, if the drop score of a particular link exceeds a fixed threshold determined from the natural packet drop rate (see Section 4.2 for details), then that link is identified as malicious. The adversarial nodes on the path may collude to share the drops amongst themselves; however in this case, the drop rate will still be bounded (proportional to the number of malicious nodes in the path).

We note that although the full-ack scheme achieves a high detection rate, it suffers from high communication overhead (see Appendix B for detailed analysis). Specifically, it requires a $\text{O}(d)$ ack-size per data packet sent out by the source, irrespective of whether the packet was dropped on the forwarding path. We note that in a real world scenario, attacks are not very frequent, and therefore, high communication overhead per packet even in the absence of an attack is unacceptable. This motivates the design of our PAI protocol that reduces the communication complexity to $\text{O}(1)$ per data packet at the cost of a decrease in the detection rate.

We note that the protocol due to Avrompoulos et al. [2, 3] would achieve a similar performance as that of the full-ack protocol described above (although the authors do not provide any concrete analysis of their protocol). However, the full-ack protocol is rather intuitive and easy to analyze; therefore we give its performance evaluation in Section 4 for comparison with our PAI protocol.

3.2 The PAI Protocol

Before we explain the details of our approach, we first present a brief overview highlighting the main intuitions behind our approach.

3.2.1 Overview

In our protocol, the base station must return an *acknowledgement* packet (ack) for each *data packet* sent by the source. If S does not receive a valid ack from the base station within a specified wait-time, it will gather evidence of where the packet was dropped on the path by sending out a probe packet towards the base station. The *probe packet* selects a node on the path uniformly at random, and the selected node must return an ack to S if it had earlier received (and forwarded) the data packet. We utilize cryptographic mechanisms to hide which node was

selected, thus, even traffic analysis does not reveal the node that generated the ack. Our approach is probabilistic in nature, and a malicious link (with excessive drop rate) will be detected after sufficient evidence is accumulated.

To identify malicious links, our protocol utilizes a scoring mechanism that works as follows (this is completely different from the "drop score" described in Section 3.1). Let E denote the event that a particular node f_i is *selected* when a packet drop occurs at its adjacent upstream link l_i during a given round. We provision that event E occurs with a fixed probability. Now, on each occurrence of event E, we assign some constant score to each link l_j ($j \in [0, i]$) upstream of f_i, while the scores of all downstream links are unaffected, thereby creating a difference in the scores of the links on either side of f_i. Over a period of time, the scores of each link are accumulated and a difference in the score of two adjacent links would indicate a potential malicious link.

To localize and identify the malicious links, we first set a threshold for the end-to-end drop rate for a given path. The threshold value is chosen according to the natural drop rate, such that the end-to-end drop rate if caused merely by the natural drop will not exceed the threshold value. At the end of each round, S computes the end-to-end drop rate so far; if it exceeds the threshold value, then it indicates that some adversary was present on the path. Using the history of scores (i.e., the scores accumulated so far) of the links, S will identify the adversarial presence on a link (or a set of links) whose score exceeds a per-link score threshold *within a bounded number of probes*. On the other hand, the score of an honest link will not exceed the per-link score threshold. Note that this mechanism is in sharp contrast to the on-demand secure routing approach [5] where the probing is launched only when the end-to-end drop exceeds a certain threshold; consequently there is no history of scores which can be used, thus allowing an adversary to freely drop packets until the end-to-end drop reached the threshold and then cause arbitrary links to be incriminated due to natural packet loss when probing is initiated. As with any ack-based scheme, we can only identify unreliable *links* since identifying the malicious node is not possible: either endpoint of a link could be the culprit.

We note that an adversary may choose to alter or drop either of the following: (a) data packet, (b) probe packet, and (c) ack packet. However, any such activity at a link has the same result as dropping the data packet at that link. From now on, we use the term "drop" to refer to any kind of packet alteration or drop. Later, in Section 4.2, we show that an adversary achieves the same total end-to-end drop rate with different individual drop rates for different packet types.

3.2.2 Protocol

The PAI protocol consists of five phases: (a) *send data and decide whether to probe*, (b) *probe*, (c) *acknowledge*, (d) *score*, and (e) *identify*. We now proceed to the description of these phases.

Phase 1: send data and decide whether to probe
Let us consider that S sends out a data packet m_j towards the base station. A data packet m is of the form:

$$m = \langle data||timestamp||\text{MAC}_{nonce}(data||timestamp)\rangle$$

where $nonce$ is a cryptographic nonce computed by using a PRF keyed with the secret key K_d shared between S and the base station in the following manner:

$$nonce = \text{PRF}_{K_d}(data||timestamp)$$

An intermediate node accepts a data packet only if the timestamp embedded in the packet is recent. We defer until Section 4.1 to explain the importance of the timestamp.

On receiving m_j, the base station computes $nonce_j$ and then verifies the correctness of the MAC value. If verification fails, it drops the received packet. Otherwise, it sends back the following ack to S.

$$\langle h(m_j)||nonce_j\rangle$$

On receiving this ack, an intermediate node f_i computes a MAC over the $\langle data||timestamp\rangle$ value embedded in the data packet m_j received earlier using $nonce_j$ as the secret key, and verifies it against the MAC value embedded in m_j. Since $nonce_j$ can only be computed by either the source or the base station, f_i can verify its correctness to decide whether m_j was received unaltered at the base station. If verification succeeds, f_i deletes the state maintained for m_j.

If S receives a valid ack from the base station within a waiting time, it concludes that m_j arrived unaltered at the base station and the protocol is terminated for the j^{th} round. Otherwise, S decides to gather evidence by sending

a probe to select a node on the path uniformly at random. This selected node must return an acknowledgement to S.

Phase 2: probe
S sends out a probe packet c_j towards the base station. The probe packet contains an identifier $h(m_j)$ for the data packet m_j and a random challenge R_j.

$$c_j = \langle h(m_j)||R_j \rangle$$

On receiving this probe packet, each intermediate node f_i must first verify the embedded hash value to check whether a corresponding data packet was received earlier. If verification succeeds, it computes a PRF-based predicate T over input R_j by using the symmetric key K_i shared with S. The predicate varies for each node f_i and returns true with some probability p_i.

Definition 1 *We say that a node f_i is **sampled** for a data packet m_j if the predicate* T *returns true for input R_j.*

Finally, f_i starts a wait-timer $t_i^{m_j} = r_i$ (where r_i is the round-trip time from f_i to the base station) and forwards the probe packet towards the base station.

Phase 3: acknowledge
In this phase, the intermediate nodes must send back an acknowledgement to the base station. We define an ack from an intermediate node with the following structure:

$$a = \langle h(m)||\mathcal{A}_i^m \rangle$$

Here, $h(m)$ serves as an identifier for data packet m_j and $\mathcal{A}_i^m$ is a report computed by node f_i for that data packet.

The ack for data packet m_j must originate at the upstream node of the link where m_j was dropped. To achieve this, we require that if an intermediate node f_i does not receive any ack from its downstream neighbor within the waiting time $t_i^{m_j}$, it must generate a new ack $a_i = \langle h(m_j)||\mathcal{A}_i^{m_j} \rangle$, where the report $\mathcal{A}_i^m$ is computed as:

$$\mathcal{A}_i^{m_j} = \mathrm{E}_{K_i}(f_i||c_j||\mathrm{MAC}_{K_i}(f_i||c_j)) \tag{1}$$

However, on receiving an ack within the waiting time $t_i^{m_j}$, an intermediate node f_i performs one of the following actions:

1. If f_i was sampled for m_j during phase 1, it generates a new ack (as described in Equation 1) to overwrite the received ack.
2. Otherwise it re-encrypts the report in the received ack, i.e., $\mathcal{A}_i^{m_j} = \mathrm{E}_{K_i}(\mathcal{A}_{i+1}^{m_j})$.

Finally, f_i forwards the ack $a_i = \langle h(m_j)||\mathcal{A}_i^{m_j} \rangle$ towards the source.

Definition 2 *We say that a node f_e is **selected** for a data packet m_j, if (a) f_e is sampled for m_j, and (b) $f_1, \ldots, f_{e-1}$ are not sampled .*

For a given data packet, only one intermediate node is *selected* uniformly at random with probability P. For an intermediate node f_i, if we set the probability that the predicate T (defined in Phase 2) returns true to be $p_i = \frac{1}{d-i+1}$, we have $P = \frac{1}{d}$, where d is the hop count between the source S and the base station.

Observe that due to the ack forwarding mechanism described above, S expects an ack packet that was generated at the *selected* node f_e and re-encrypted by *each* upstream node between f_e and S.

Phase 4: score
On receiving an ack, S first executes a decoding algorithm to verify its contents. The algorithm is explained below.

1. The input to the algorithm is the report $\mathcal{A}_1^{m_j}$ in the ack received from f_1. However, if no ack is received within the wait-time, the input is set to *null*.
2. The algorithm decodes the input by performing successive decryptions using the keys $K_1, \ldots, K_e$ *in that order*, where K_e is the symmetric key shared between S and f_e. If the final decoded value is the same as the expected value $\langle f_e||c_j||\mathrm{MAC}_{K_i}(f_e||c_j) \rangle$, the algorithm outputs *true*, otherwise *false*.

Based on the output of the algorithm, S assigns numerical *scores* to the links. These scores are accumulated over a period of time and eventually, the adversarial links are identified based on their scores. The scoring procedure works as follows.

1. If the algorithm returns *false*, then S is convinced that *there exists at least one malicious link* in the interval $[l_0, l_{e-1}]$. Since each link in this interval has equal probability of being malicious, S adds 1 to the individual score of each link. The scores of each link in the interval $[l_e, l_{d-1}]$ is unaffected.

2. In contrast, when the algorithm returns *true*, S is convinced that there was no malicious activity in the interval $[l_0, l_{e-1}]$. Therefore, no score is added for any link.

We now provide some intuition for our score-based identification mechanism. Suppose that link $l_{\mathcal{M}}$ is malicious. Let E be the event that $f_{\mathcal{M}+1}$ is *selected* when a drop occurs at $l_{\mathcal{M}}$ during a round. Observe that E occurs with a fixed probability $\frac{1}{d}$. On each occurrence of E, the score of each upstream link l_i, $i \in [0, \mathcal{M}]$ would be increased by 1, while the scores of all downstream links $l_{\mathcal{M}+1}, \ldots, l_{d-1}$ would remain unaffected, thereby creating a difference of 1 in the scores of $l_{\mathcal{M}}$ and $l_{\mathcal{M}+1}$. As the scores of each link are accumulated over a period of time, a difference in the score of two adjacent links would indicate a potential malicious link.

Phase 5: identify
At any point of time, let s_i be the score of link l_i. We define δ_i as the score difference $s_{i+1} - s_i$ of two adjacent links l_i and l_{i+1}. As we show later in Section 4.2, the value of δ_i directly depends on the drop rate of each link l_j, where $j \leq i$.

We first set a *per-link threshold*, denoted by α to determine whether a *particular link* is malicious or not. The value of α should be set close to the maximum value of the natural per-link drop rate to ensure that the drop rate of an honest link does not exceed α while at the same time guaranteeing a desirable end-to-end throughput .

We now compute a threshold ψ_{th} for the end-to-end drop rate by using α as the per-link drop rate. ψ_{th} will guarantee that: (a) if the actual drop rate at none of the links exceeds α, the actual end-to-end drop rate will not exceed ψ_{th}; (b) if the actual end-to-end drop rate exceeds ψ_{th}, there must be *at least one* malicious link with drop rate larger than α.

The identification mechanism works as follows. At the end of phase 4, S computes the new value of the end-to-end loss rate accumulated so far, and checks whether it exceeds the threshold ψ_{th}. If the check fails, S keeps all the scores intact and returns to Phase 1. Otherwise, S proceeds to identify the malicious links. It first computes the individual drop rate θ_i for each link l_i. Note that, with the knowledge of δ_i and the count of the number of times that f_i was *selected*, S can determine θ_i. S then compares the value of each θ_i with α; if $\theta_i > \alpha$ then l_i is identified as malicious. At this stage, S resets accumulated end-to-end drop rate and the scores of each link. S can now re-route around the links identified as malicious. Note that it is possible that certain malicious links drop packets with rates higher than α while the end-to-end drop rate does not exceed ψ_{th}. However, in this case the end-to-end drop rate is still bounded by the threshold ψ_{th} (thus we do not consider this as false negative). We defer until the next section to show how each value described above is computed, and in order to reduce the detection false positive (a honest link may temporally exhibit an average drop rate more than α due to the estimation variance), S would need to wait for a *constant* number of packet transmissions from the source (see Theorem 4) before identifying malicious links according to α.

4 Analysis of PAI

We begin this section by describing the important properties of our protocol. Next, we present the security bounds for our protocol. In Appendix B we give an analysis for storage and computation overheads of PAI protocol.

4.1 Protocol Properties

Here we describe the main properties of our protocol and explain why they are essential to prove the security in our adversary model.

Property 1: Adversary localization. First recall from Phase 3 of the protocol that a *sampled* node must *overwrite* the ack received from a downstream node with a fresh ack. For a given data packet, although more than one node may be sampled, the final ack packet received at the source is the one that was generated by the *selected* node. Therefore, if the *selected* node is located between the source and the adversary, then the adversary cannot

influence the final ack packet received at the source. This implies that if no ack or an invalid ack is received at the source, then there must exist at least one malicious link in the interval $[l_0, l_{e-1}]$.

Property 2: Delayed sampling. Since a sampled node must generate a fresh ack, we note that if a malicious node could determine whether it is sampled for a given data packet immediately when it receives the data packet, it could safely drop the data packet without increasing its probability of being identified. Therefore, we use a delayed sampling mechanism, where a probe packet c_j is sent at a later time to sample the nodes for a data packet m_j sent at an earlier time. Now, a malicious node may try to wait for the arrival of c_j before sending m_j, in an attempt to decide whether or not to forward m_j. Therefore, we require loose time-synchronization amongst the nodes in the network such that the clock error between two adjacent nodes f_i and f_{i+1} is less than $min(r_0)$, i.e. the minimum value of the round trip time from S to the base station. In this scenario, an intermediate node would discard a data packet that carries an expired timestamp.

Property 3: Ack indistinguishability. Note that in our adversary model, an attacker can perform traffic analysis at any link in the network. Now recall the ack forwarding mechanism described in Phase 2 of the protocol. Note that if an intermediate node does not receive any ack within a wait-time, it generates a new ack even if it is not sampled; otherwise an adversary could observe the ack origin to infer whether an intermediate node is sampled. Further, the *re-encrypt or overwrite* technique ensures that a constant size ack is forwarded at each hop. If this were not the case, then an adversary who eavesdrops at all the links on the path to observe any difference in the size of the ack at various links can infer additional information about the origin of the ack.

For a given data packet, the probed node is *selected* uniformly at random. Note that if this were not the case, then an adversarial node could simply preferentially perform dropping attacks at nodes that are not as likely to be sampled as others.

4.2 Security Bounds

Let ρ_i be the natural drop rate of link l_i, and suppose that ρ_is are i.i.d. random variables with expectation ρ. Let α denote the per-link drop rate threshold; and θ_i be the actual average drop rate of link l_i, including both natural and malicious drops. Let η_i be the number of times that node f_i is *selected* so far. Denote by ψ_d the actual end-to-end drop rate and by ψ_{th} the end-to-end drop rate threshold. Let ζ be the malicious end-to-end drop rate, i.e., the drop rate due to the malicious links. We first assume that the adversary employs identical drop rate for all kinds of packets (data, probe or ack packet), and thus the probability that a packet of any kind is dropped at l_i is θ_i. Later in Theorem 3, we show that by varying the drop rates for different kinds of packets, an adversary still achieves the same maximum end-to-end drop rate. When the observed drop rate value converges to its true value within a small uncertainty interval, the detection false positive is limited below a certain threshold ϵ. We call this the *converged* condition. For the ease of understanding, all the analysis results before Theorem 4 are for the converged condition. In Theorem 4 we derive the detection rate for PAI, compared to the "optimal" detection rate of the full-ack scheme (Section 3.1). All the corresponding proofs are given in Appendix A.

Lemma 1 *The score difference $\delta_i = s_{i+1} - s_i$ is given by $\delta_i = \eta_{i+1} \cdot \{1 - [\prod_{y=0}^{i}(1-\theta_y)]^3\}$ where η_{i+1} is the number of times that f_{i+1} is selected, and θ_i is the average drop rate of link l_i.*

Lemma 2 *Based on the values of δ_i and η_i known to S, S can derive the average drop rate θ_i by:*

$$\theta_k = \begin{cases} 1 - (1 - \frac{\delta_0}{\eta_1})^{\frac{1}{3}}, & k = 0 \\ \theta_k = 1 - [\frac{1}{C}(1 - \frac{\delta_k}{\eta_{k+1}})]^{\frac{1}{3}} \text{ where } C = \prod_{y=0}^{k-1}(1-\theta_y)^3, & k \geq 1 \end{cases}$$

Theorem 1 ***Bounding Malicious End-to-End Drop Rate.*** *Suppose the network is under a converged condition. Given the per-link drop rate threshold α, path length d, and the number z of malicious links in the given path, by setting the end-to-end drop rate threshold ψ_{th} as $\psi_{th} = 1 - (1-\alpha)^{2d}$, an adversary in control of z malicious links (or z adversarial links that collude) can cause at most $\zeta = 1 - \frac{(1-\alpha)^{2d}}{(1-\rho)^{2(d-z)}}$ end-to-end drop rate without being detected.*

From Theorem 1 we can see that the location of the malicious links on the path exercises no influence on the end-to-end drop rate. We can further derive the relationship between the malicious end-to-end drop rate ζ and natural loss rate ρ as well as the number of malicious links z, as Theorem 2 shows.

Theorem 2 *Given a fixed number of malicious links, the malicious end-to-end drop rate ζ increases approximately linearly with the increase of natural loss rate ρ. Given a fixed number z of malicious links, the optimal strategy for the adversary in order to cause the maximum ζs across all the paths containing malicious links in the sensor network is to deploy one malicious link for one path. In this case, the total malicious drop rate across all paths containing compromised links increases linearly with z.*

For a path of length $d = 6$, $\rho = 0.5\%$, $\alpha = 1\%$, and with one malicious link, the adversary can drop at most around 7% traffic on one path without being detected. Figure 3(a) in Appendix A plots the overall fraction of traffic dropped by the adversary across all paths containing malicious links against α and z (where "on one path" refers to the scenario where z malicious links exist on one path). Figure 3(b) plots the fraction of traffic dropped by the adversary on one path against various path lengths with a single malicious link.

In Theorem 1 we assumed that the adversary drops all kinds of packets at an identical drop rate. However, the adversary may choose to drop different types of packets with different probabilities. In the following theorem we show that given a fixed α (thus a fixed ψ_{th}), an adversary will achieve the same maximum end-to-end drop rate with different drop rates for different packet types.

Theorem 3 *An adversary who employs different drop rates for different types of packets achieves the same maximum end-to-end drop rate given the per-link drop rate threshold α.*

Now we compute the detection rates of the full-ack scheme in Section 3.1 and PAI in the following theorem.

Theorem 4 *Given the threshold $\alpha = \rho + \epsilon$ and the allowed false positive σ, the full-ack scheme requires $\tau_1 = \frac{ln(\frac{2}{\sigma})}{8\epsilon^2 \cdot (1-\rho)^{2+d}}$ packets transmitted by the source to converge; while PAI requires at most $\tau_2 = 2^d \frac{ln(\frac{2}{\sigma})}{18\epsilon^2 (1-\rho)^{\frac{4}{3}}} \cdot d \cdot log(d)$ packets to converge.*

For example, with $\epsilon = 0.03$, $\sigma = 0.03$, $\rho = 0.02$, and $d = 5$, we have $\tau_1 \doteq 7 \times 10^2$ and $\tau_2 \doteq 3 \times 10^4$. Note that the detection rate of PAI degrades with larger value of d, thus PAI is most efficient to detect malicious links that are close to the base station. We note that in a practical scenario, an attacker (if present) would tend to compromise links close to the base station that aggregate more traffic than the *edge* links.

5 Bounding Message Injection Attacks

The counterpart of a packet dropping attack is a packet injection attack, where a node inside the network originates a large amount of communications to drain the resources of legitimate nodes in the network. More insidiously, the adversary could send large volumes of data down specific links or paths, crowding out legitimate data. If this occurs, then a packet-dropping detection mechanism may falsely identify these crowded links as malicious. Clearly, for a bound on packet dropping to be meaningful, the adversary's capability to perform packet injection must also be bounded.

In this section, we describe a countermeasure against a packet-injection attacker. Our approach is to perform communications in epochs. Within each epoch, each node is allotted a fixed maximum rate of message origination. This is the upper bound on data injection that we wish to enforce at each node in the network, and is selected by the operator of the network as an input to the algorithm (note that different nodes may have different bounds). When each epoch is complete, the base station then audits the total amount of communications of each node in the network. If any node in the network exceeds its allotted rate, the adversary will be unable to pass the audit and will have to pay a penalty; typically this means some set of edges or nodes under its control will be revoked for subsequent epochs.

5.1 Problem Definition

Informally, we wish to detect if the adversary has caused a large amount of traffic to be carried by the legitimate nodes in the network. We define the problem formally as follows.

We associate with each node a, a maximum data origination rate R_a which represents its per-epoch origination bandwith allotment. In a given epoch, each link e in the topology carries some amount of traffic destined for the base station. Let b_e denote the total amount of traffic that traversed link e within the epoch. Visualising the set of b_e as weights on edges, the weight assignment for a given epoch is called the *communication pattern* of the epoch:

Definition 3 A **communication pattern** is an assignment of some traffic volume b_e for each link e in the network.

A communication pattern where every node conforms to its rate limits is called "well-behaved", defined as follows:

Definition 4 A **well-behaved communication pattern** is a communication pattern in which, for every node a in the network, the total amount of outgoing traffic is no more than the sum of the total amount of incoming traffic and the node's maximum data origination rate R_a.

Note that the definition allows for nodes to have less outgoing traffic than incoming traffic, reflecting some packet loss at that node. Intuitively, we would like to allow only well-behaved communication patterns in each epoch; however such a security definition is too strict. For example, if two malicious neighboring nodes send an illegally large volume of traffic to each other, this traffic is invisible to the legitimate nodes in the network and yet the communication pattern would violate the "well-behaved" constraint. Hence we offer a more functional definition in terms of only what is visible to the legitimate nodes.

Definition 5 The **view** of the legitimate nodes is an assignment of some traffic volume b_e for each link e in the network which is adjacent to a legitimate node.

Since the view only assigns weights to a subset of the links in the network, we can consider if it is in fact possible to assign weights to the remaining links such that no node in the network exceeds its allotted rate. We formalise this approach as follows:

Definition 6 For a given view W, let F be any communication pattern which assigns the same weight as W to any edge in W. Then F is a **completion** of W. A **permissible** view W is one where there exists some well-behaved communication pattern which is a completion for W.

For a given epoch, if the view is permissible, then the existence of the well-behaved completion implies that exists some way for this view to have occured legitimately. In other words, the adversary could have achieved this amount of traffic on the legitimate nodes simply by obeying the rate limits on the nodes under its control. On the other hand, if the view is not permissible, then any communication pattern C which results in the view V must not be well-behaved: in particular, there must always be at least one node in the network which exceeds its maximum allowed data origination rate. Hence, we can say that the network is under data origination attack if and only if the view of the legitimate nodes is not permissible.

5.2 Direct Audit

Based on the problem definition, the most direct method for determining if the view of the legitimate nodes is permissible is to check the amount of traffic across every link in the network. The base station can then directly check if the reported traffic weights reflect a well-behaved communication pattern.

The details of this direct audit process is as follows. At the end of each epoch, the base station broadcasts a end-of-epoch message containing the epoch identifier N_e. Every node a then composes a *traffic report* D_a of the total communications it has performed over the epoch on each of its incident links. The message is encrypted and authenticated using the secret key K_a that a shares with the base station.

Once all the traffic reports are received, the base station can then take the union of the reports to find the communication pattern. To resolve conflicting reports on the amount of traffic on a given link, we note that if link-layer encryption and authentication is implemented, an adversary cannot inject messages between legitimate nodes. Hence, for any link e from node c to node p, the downstream node must have received no more messages than was sent by the upstream node. The base station checks that the respective traffic reports reflect this fact, then assigns the traffic weight reported by p to the link. When each edge has been labelled with a traffic weight, the base station checks the well-behavedness property of the overall communication pattern C': specifically, for each node i, the amount of outgoing traffic from i must be no more than its maximum origination rate R_i more than the total amount of incoming traffic to i.

Theorem 5 *If C' is well-behaved, then the view of the legitimate nodes is permissible.*

Essentially, C' assigns the same traffic weight as the legitimate nodes to all edges in the legitimate view except it may assign smaller weights on outgoing edges from legitimate nodes. This discrepancy does not help the adversary exceed its rate. A proof of the theorem is in the Appendix C.

5.3 Locating an Adversary

In the case of a failed audit, we would like to be able to locate at least one link or one node under adversarial control and revoke it, thus applying some kind of penalty to the adversary for the attack. Thus, if the adversary persists in performing such attacks, eventually all its points of presence in the network will be located and revoked.

The detection mechanism of Section 5.2 ensures that if an adversary performs an attack in an epoch, it cannot pass the requisite checks. However, rather than incriminate itself by revealing that one of its nodes is not obeying its rate limit, the adversary can simply refuse to respond to the audit. More importantly, it can choose to selectively drop the reports of various legitimate nodes, thus denying the base station any information which can be used to locate the adversary's point of presence in the network.

We modify the protocol to prevent arbitrary dropping of traffic reports. First, we select a subset of the network's edges that form a spanning tree directed towards the root (the base station). We assume that certain relevant details of the topology of the spanning tree are known to the nodes; in particular, each node is aware of its set of children in the spanning tree, and also the size of the subtrees associated with each child. Since this spanning tree is largely static, such details could be sent from the base station to each node on the initialisation of the network, and changes can be disseminated efficiently using authenticated broadcast to the affected subtrees.

When a node a receives the end-of-epoch message, it waits to receive traffic report messages from its children. The receiving node a checks the traffic reports for size (the report from a child c_i should be at most proportional in size to the number of nodes in subtree rooted at the c_i). If the traffic report is too large, a refuses to accept the message. This prevents the adversary from using the audit phase as an opportunity to flood the network with a large volume of traffic. Once a has received the reports from all its children, it then concatenates all the received information with its own traffic report into a single message, which secured with the MAC using the key K_a shared between a and the base station. This message is then sent to the parent of a which then repeats the process. Formally, this process is described as follows, where $c'_i, \ldots, c'_r$ are the children of a in the spanning tree and D_a is the traffic report of node a:

$$M'_a = D_a||M_{c'_1}||\cdots||M_{c'_r}, \quad M_a = M'_a||\text{MAC}_{K_a}(M'_a)$$

This method of repeated encapsulation ensures that a malicious adversary cannot selectively drop the reports of nodes at arbitrary points in the network. The adversary can only drop the messages of an entire subtree in the spanning tree; furthermore it can only do this if it controls the link between the root of the subtree and its parent (for example, one of the two endpoints of the link may be malicious). Hence, by observing which subtrees did not successfully transmit their traffic reports, the base station can determine which communication links are under adversary control. The base station may then revoke these links in a subsequent epoch.

Specifically, in a given epoch where the audit fails, the base station revokes any link e between parent node p and child node c if (a) p claims to have received more messages than c claims to have sent out or (b) e was a spanning tree edge and no traffic report was received from c. We assume that during the audit phase we can allow for a sufficiently large number of retransmission attempts such that a link between two legitimate nodes should eventually succeed in delivering the message in the absence of adversarial interference.

5.4 Bounds

We have established that if the network passes a full audit, then the adversary cannot have performed an attack in that epoch. It remains to show that if the adversary decides to take an opportunity to perform an attack (and thus deliberately fail the audit), the total amount of damage can be bounded.

We label each communication link in the network with a *capacity*, which represents the maximum amount of data that can be sent over that link in an epoch. Each parent node is aware of the capacity of each of its incoming links; any data in excess of this rate is discarded by the parent.

Based on this capacity, a set of adversary nodes can inject at most some volume of data F into the network. Each time the adversary fails the audit, it can inject at most F data but loses one of the edges under its influence. Hence if E_m is the set of all edges incident to any malicious node then the total amount of extra data that can be injected in a series of failed audits is at most $|E_m|F$.

6 Distributed Detection

The detection algorithm of Section 5.2 allows us to determine if the view of the legitimate nodes is not permissible, but it requires the transmission of n traffic reports to the base station causing $O(n)$ congestion at the nodes closest to the base station even when no traffic injection attack is taking place. We describe a method for detecting

non-permissible views with $O(dn_a)$ congestion and memory overhead (where d is the maximum indegree in the network and n_a is the maximum number of ancestors for any node).

Our algorithm is motivated by the observation that the problem of auditing incoming and outgoing traffic is closely related to the problem of secure SUM aggregation. We chose to modify the secure SUM algorithm of Chan et al. [7] to perform the distributed audit for rate monitoring.

Two important complications arise making the adaptation non-trivial. First, in the SUM aggregation problem, only the final sum computed at the base station needs to be correct; however for the rate monitoring problem the subsums at every intermediate node must also be correct. We address this by making the capacity of the links incident to each node known to all its children; effectively allowing us to perform the rate consistency check at every node instead of just at the base station. The second complication is that, in the SUM aggregation problem, aggregation is performed with respect to a spanning tree; on the other hand rate-rate monitoring needs to consider the topology of the entire DAG network since traffic can flow along any edge. To address this problem we require each node to check the consistency of the operations at each one of its parents rather than just a single path.

6.1 Assumptions

To provide the properties that are necessary for the correctness of the distributed audit scheme, we make two additional assumptions, as follows:

Every node knows the full details of the topology between itself and the base station. Specifically, for a node x, let A_x denote the set of nodes on any path between x and the base station. We will call A_x the set of all ancestors of x. For any node $a \in A_x$, the node x knows every edge incident to a as well as its associated capacity. Furthermore x knows the rate limit R_a of a. Downstream topology knowledge is already implied to some extent in Section 3, since each node needs to have sufficient information to perform source routing to avoid bad links. This information can be bootstrapped using an authenticated broadcast when the network is first initialised; changes in the topology (which should be infrequent) can be disseminated using broadcasts that are limited in scope to the descendants of any affected nodes.

Capacities are assigned conservatively. In a distributed audit setting, malicious nodes without legitimate descendants are never audited. To prevent them from exceeding their rate limits, we need to directly restrict their ability to inject data by not providing them with edges of large capacity into the network. Specifically, we require that, for each node, the total capacity on its outgoing edges should be no more than the total capacity on its incoming edges, plus the origination rate of the node itself. For example, if a node has two incoming edges with capacities of 2 and 3, and it has an allowed origination rate of 1, then its total outgoing capacity should be no more than 2+3+1=6.

6.2 Algorithm Overview

The distributed audit algorithm is related to the secure SUM algorithm described by Chan et al [7]. Specifically, it proceeds in three phases: commitment, verification and confirmation. We describe each in turn.

Commitment phase. In the first phase of the algorithm, each node commits to the details of its own traffic report. The commitment structure is constructed such that each node can only commit to a single traffic report; hence a malicious node is unable to selectively reveal different traffic reports to different nodes in the hope of evading detection.

The commitment structure is a *topological hash DAG*, which is a natural generalisation of a *hash tree*. Consider a network topology in the form of a directed acyclic graph (DAG) with a single sink (i.e., the base station) which we call the root. Suppose each node x is associated with some (not necessarily distinct) label v_x (in our algorithm, the label v_x represents the traffic report of node x). An example of this network topology is shown in Figure 2.

The topologicial hash DAG is constructed recursively. An example is shown in Figure 2. First the nodes in the network which have no children set $M_x = h(x||v_x)$ and report their respective M_x to all of their parents. Internal nodes then compute hash values recursively. In the following formula, a is the internal node computing its value, and it has children $c_1, \ldots, c_k$ in order of increasing ID.

$$M_a = h(a||v_a||c_1||M_{c_1}|| \cdots ||c_k||M_{c_k})$$

The root value M_r is computed in the same way. Considering Figure 2, note that the structure of the hash DAG follows the network topology exactly. For a given M_r, it is computationally infeasible for an adversary to find another topology or assignment of values to node labels that will create a hash DAG with the same M_r.

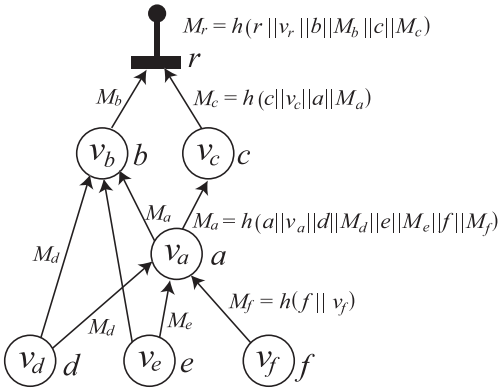


Figure 2: Topological Hash DAG

M_r is sent by the base station to the entire network using authenticated broadcast. Nodes then relay the necessary information for each other to verify their presence in the topological hash DAG. Specifically, each node a sends all the hash inputs needed for the computation of M_a to all of its descendants. Each node thus receives the inputs to the hash computation for each one of its ancestors. The dissemination process causes $O(N_a)$ congestion on the links of the network where N_a is the number of ancestor nodes for the node with the greatest number of ancestors.

Note that an adversary is unable to perform additional data injection while the hash inputs are being disseminated. This is because each node, knowing the exact downstream topology of itself and its parent nodes, is able to calculate the exact size of the data it is expecting to receive. Any additional data is simply dropped.

Verification Phase. Once a node x has received the inputs to the hash computations of each of its ancestors, it may perform verification to ensure that none of its ancestors exceeded their respective rates. First, node x repeats the relevant computations to verify that the received values do indeed result in the correct M_r after repeated hashing, and the reconstructed structure of the topological hash DAG exactly matches the known downstream topology of x. Once x is certain that the received data was indeed the same values that resulted in the commitment M_r, it then checks the consistency of the traffic reports of each ancestor node a. Specifically, x checks that (a) no node exceeded the capacity on its links; (b) for any link $e = (c \rightarrow p)$, the downstream node p does not report that more packets were received than the upstream node c claimed to have sent; and (c) none of the traffic reports show an ancestor node originated more than its allowed rate of traffic in an epoch. Node x only passes the verification process if all of the above conditions are met.

Confirmation phase. In the final phase all nodes proceed to notify the base station of their respective successes in verification. The notification from node a is in the form of the release of a special OK value $V_a = \text{PRF}_{K_a}(ID_a||OK||N_e)$ where N_e is the epoch identifier. These values are efficiently aggregated in the following way. Similar to Section 5.3, we assume the existence of a spanning tree such that each node knows its own parent in the tree. First, the leaf nodes in the tree send their OK values (if any) to their parents. Once each node has heard the aggregated OK values from each of its children, it aggregates all the received values along with its on OK value using XOR. Specifically, for a successfully verifying node a with children $c_1, \ldots, c_k$, it computes the value:

$$V_a = \text{PRF}_{K_a}(ID_a||OK||N_e) \oplus V_{c_1} \oplus \cdots \oplus V_{c_k}$$

When the base station receives all the aggregated values it can recompute the relevant PRF computations to check that every node has successfully verified the correctness of the traffic reports. If so, then no nodes need to be revoked; if not, then an adversary is present somewhere in the network. The base station then runs the detection protocol of Section 5.3 to identify some edge of the adversary to revoke.

6.3 Correctness

We show that, if all the legitimate nodes successfully perform distributed verification, then the view of the legitimate nodes must be permissible. For the remainder of this subsection, assume that every legitimate node has successfully performed distributed verification; our goal is to exhibit a well-behaved communication pattern as a completion to the view of the legitimate nodes. All the corresponding proofs are given in Appendix C.

Lemma 3 *Let a be any common ancestor of any two legitimate nodes i, j. Then it must provide the same traffic report to both i and j.*

Lemma 4 *Consider a DAG network where edge capacities are assigned conservatively, Consider a set of nodes $U = \{v_1, \ldots, v_k\}$. Let $S(U)$ be the set of U and all descendant vertices of any vertex in U. Then, for any assignment of (traffic) weights on the outgoing edges of $S(U)$ that respect the capacities of those edges, there exists an assignment of traffic weights on the internal edges of $S(U)$ which respects edge capacities such that no node in $S(U)$ exceeds its allotted origination rate limit.*

Theorem 6 *If all legitimate nodes pass the distributed verification, then the view of the legitimate nodes must be permissible.*

7 Combined analysis

In both the PAI and rate-limiting protocols, misbehavior in a single period implies identification of an adversary link with high probability. This allows each protocol to complement the other in ensuring data delivery and bounding the total misbehavior of the adversary. For example, an adversary may attempt to deliberately induce false positives in the packet detection protocol and thus mask the location of the actual misbehaving malicious nodes in the network. If the adversary uses packet injection to induce faults in legitimate nodes, the rate-limiting mechanism will detect this and cause a revocation of some malicious link. Hence, assuming that the PAI protocol identifies roughly one bad link per epoch of the rate-limiting protocol, even if the attacker tries to escape the packet-dropping detection mechanism, it will be caught by the packet injection mechanism and made to pay the same price (loss of one link). On the other hand, an adversary may attempt to circumvent the rate-limiting mechanism by dropping or altering legitimate packets thus causing legitimate nodes to expend their allotted origination rates through a large number of retransmissions. However, with the PAI protocol in place, any malicious edge which performs such an attack will cause its children to route away from it in the next epoch. In this sense packet-injection and packet-dropping form an overlapping defense, covering each other's weaknesses.

Let the amount of legitimate traffic per epoch be ω, and assume that one round of the PAI protocol occurs per epoch of the rate-limiting protocol. Under the rate-limiting mechanism, in the worst case an adversary could flood the network such that all legitimate traffic in an epoch is lost. However, such an attack costs the adversary one malicious link. Hence, the total amount of traffic disrupted by the adversary is $|E_m|\omega$ where E_m is the set of all links under malicious control. Under the PAI mechanism, each malicious link can perform at most one epoch's worth of packet drops for each legitimate node before that node reroutes around it. Summing over all legitimate nodes for every malicious edge, the total amount of packet drops is also at most $|E_m|\omega$. Combining the two bounds, the total amount of *detectable* misbehavior that an adversary can commit is at most $2|E_m|\omega$. This is a fixed constant which is small compared to the total traffic carried by the network in its lifetime. In other words, the best strategy for an adversary is to constrain its misbehavior such that it is unlikely to be detected and revoked, to ensure its longevity in the network for the greatest amount of damage.

8 Related Work

To the best of our knowledge, no previous work attempts to secure the data plane by addressing the problems of packet dropping and injection in conjunction. Here, we discuss related works that address either problem separately.

Packet Dropping. To the best of our knowledge, no significant attempt has been made to adress packet dropping attacks in sensor networks. McCune et al. [12] address denial of message attacks on sensor network broadcasts; however, their protocol is limited to the detection of an attack (rather than the location of the adversary) and does not consider *unicast* communication mode. We now discuss the previous solutions to packet dropping attacks in the Internet and ad-hoc networks.

Perlman [15] first introduced the idea of using an acknowledgement based scheme to detect Byzantine adversaries at the network layer. However, their protocol can be easily defeated in the presence of multiple adversarial nodes that collude to incriminate honest links as malicious. Several new protocols were proposed that build upon the ack-based approach. Unfortunately, all these protocols either incur high communications overhead, or fail to achieve security in the presence of colluding adversaries and natural packet loss. Awerbuchet al. [5] propose an on-demand routing protocol for ad-hoc networks that detects packet forwarding faults and routes around them.

They consider an end-to-end loss metric and trigger a probing mechanism only when this metric exceeds a certain threshold. However, this allows an adversary to *freely* drop packets until the metric reaches the known threshold value. Now, when probing is initiated, arbitrary honest links can be incriminated due to natural packet loss. Padmanabhan and Simon present a secure traceroute protocol [14] for the Internet that utilizes a cumulative ack based approach based on secret sharing. However, we note that an adversary can incriminate an honest link by selectively dropping the acks from a particular downstream intermediate node. Moreover, as the authors admit, traceroute is rather expensive and its use should be limited whenever possible. Liu et al. [10] propose an ack-based protocol that divides a given path into a set of triples of nodes, where one node monitors the link between the other two nodes. While this approach reduces the communication complexity of the protocol, it also in turn allows colluding malicious nodes to evade detection or incriminate honest links as malicious.

Avrampoulos et al. [2, 3] present a routing protocol by combining techniques such as source routing, hop-by-hop authentication and ack-based probing. However, for a *given source-destination pair*, their scheme induces a key storage overhead of $O(d^2)$ and a communication overhead equivalent to that of the strawman approach described earlier (see Sections 3.1 and 4). Avramopoulos and Rexford [4] propose a stealth probing protocol that utilizes an IPsec tunnel to transmit packets between a given source destination pair. Clearly, using such encryption tunnels is infeasible for a sensor network environment.

Bradley et al. [6] introduced a distributed monitoring approach to detect malicious routers by using a traffic validation mechanism based on the law of conservation of flow. However, their scheme is impractical in practice as later demonstrated by Hughes et al. [8]. Recently, Mizrak et al. [13] and Argyraki et al. [1] improve upon the scheme of Bradley et al. [6]. However, we note that these schemes utilize special features such as link state routing which work well in the Internet, but may not be applicable to the sensor networks due to incompatibility with sensor network architecture and high overhead.

Packet Injection. There are two aspects to packet injection attacks in a sensor network environment: (a) a compromised sensor node may send out *false* sensor readings to prevent correct functioning of the network, or (b) it may inject a large number of packets to crowd out legitimate traffic or cause legitimate nodes to exhaust their battery. Previous solutions address the former category of packet injection attacks [18, 19] in sensor networks. However, in this paper, we are interested in attacks where an adversary injects a large amount of spurious traffic to exhaust the battery of legitimate nodes, or crowd out legitimate traffic. Wood et al. [17] and Karlof et al. [9] summarized such attacks in sensor network environments in their survey paper.

9 Conclusion

We present two protocols for addressing the two main subcomponents of data plane security: packet dropping and packet injection. To bound packet dropping, we implement a probabilistic acknowledgement scheme using an indistinguishable probing process. Our PAI protocol can bound the end-to-end drop rate below a fixed threshold for a given path in the presence of multiple adversarial nodes and natural packet loss. To bound packet injection, we perform an epoch-based audit of the total amount of traffic across each link to determine if any node is exceeding its allotted rate. If so, then at least one adversary edge is identified and revoked. The resultant bound on the attacker's packet injection power is $|E_m|F$ where E_m is the set of edges incident to the set of malicious node and F is the total flow that the malicious nodes can send by disregarding their rate limits. We note that for a fixed number of adversary nodes in the network, the total amount of misbehavior (packet injection or dropping) is a constant regardless of the lifetime of the network. This is the first comprehensive bound on data plane misbehavior that has been established by any scheme in the literature.

References

[1] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and delay accountability interface for the internet. In *Proceedings of IEEE International Conference on Network Protocols*, 2007.

[2] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Amendment to: Highly secure and efficient routing. Available at `http://www.princeton.edu/~iavramop/amendment.pdf`.

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

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

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

[6] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 115–124, Oakland, CA, May 1998.

[7] H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation for sensor networks. In *Proceedings of the ACM Conference on Computer and Communications Security*, Oct. 2006.

[8] J. R. Hughes, T. Aura, and M. Bishop. Using conservation of flow as a security mechanism in network protocols. In *Proceedings of IEEE Symposium on Security and Privacy*, 2000.

[9] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *Elsevier's AdHoc Networks Journal, Special Issue on Sensor Network Applications and Protocols*, 1(2–3):293–315, Sept. 2003.

[10] K. Liu, J. Deng, P. K. Varshney, and K. Balakrishnan. An acknowledgement-based approach for the detection of routing misbehavior in MANETs. *IEEE Transactions on Mobile Computing*, May 2007.

[11] D. Malan, M. Welsh, and M. Smith. A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography. In *Proceedings of IEEE Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, Oct. 2004.

[12] J. McCune, E. Shi, A. Perrig, and M. K. Reiter. Detection of denial-of-message attacks on sensor network broadcasts. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2005.

[13] A. T. Mizrak, Y.-C. Cheng, K. Marzullo, and S. Savage. Fatih: detecting and isolating malicious routers. In *Proceedings of International Conference on Dependable Systems and Networks*, 2005.

[14] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM Computer Communication Review (CCR)*, 33(1):77–82, 2003.

[15] R. Perlman. *Network Layer Protocol with Byzantine Agreement*. PhD thesis, The MIT Press, Oct. 1988. LCS TR-429.

[16] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept. 2002.

[17] A. Wood and J. Stankovic. Denial of service in sensor networks. *IEEE Computer*, pages 54–62, Oct. 2002.

[18] F. Ye, H. Luo, S. Lu, and L. Zhang. Statistical en-route filtering of injected false data in sensor networks. In *IEEE Infocom*, 2004.

[19] S. Zhu, S. Setia, S. Jajodia, and P. Ning. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *Proceedings of IEEE Symposium on Security and Privacy*, 2004.

A PAI Protocol Analysis: Proofs

Proof of Lemma 1: δ_i will be *increased by 1* **iff** the following event occurs: (a) f_{i+1} is *selected* and (b) f_{i+1} fails to ack. Define ψ_{i+1} as the conditional probability that f_{i+1} fails to ack given that f_{i+1} is *selected*. Note that f_{i+1} sends an ack successfully only if all of the following three packets are successfully delivered: (a) data packet from S to f_{i+1}, (b) probe packet from S to f_{i+1}, and (c) ack from f_{i+1} to S. Either of these three has the same following probability for successful delivery:

$$\prod_{y=0}^{i}(1-\theta_y)$$

Therefore, we have:

$$\psi_{i+1} = 1 - [\prod_{y=0}^{i}(1-\theta_y)]^3$$

Given that f_{i+1} is *selected* at η_{i+1} times, the increase of δ_i is given by:

$$\delta_i = \eta_{i+1} \cdot 1 \cdot \psi_{i+1} = \eta_{i+1} \cdot \{1 - [\prod_{y=0}^{i}(1-\theta_y)]^3\}$$

This proves the lemma. ■

Proof of Lemma 2: We prove this by mathematical induction as follows.
(a) We can compute θ_0 from the following:

$$\delta_0 = \eta_1 \cdot [1-(1-\theta_0)^3] \;\; \Rightarrow \theta_0 = 1 - (1 - \frac{\delta_0}{\eta_1})^{\frac{1}{3}}$$

S knows the values of δ_0 and η_1, so it can compute δ_0.
(b) Suppose the values of $\theta_0, \theta_1, \ldots, \theta_{k-1}$ have been derived. Then we have:

$$\delta_k = \eta_{k+1} \cdot [1 - C \cdot (1-\theta_k)^3] \;\; \Rightarrow \theta_k = 1 - [\frac{1}{C}(1 - \frac{\delta_k}{\eta_{k+1}})]^{\frac{1}{3}}$$

where $C = \prod_{y=0}^{k-1}(1-\theta_y)^3$ can be computed with the values of $\theta_0, \ldots, \theta_{k-1}$. Since S also knows the values of δ_k and η_{k+1}, it can compute δ_k. This proves the lemma. ■

Proof of Theorem 1: We first need to determine the end-to-end drop rate threshold ψ_{th}. It must ensure that, the actual end-to-end drop rate exceeds ψ_{th} only when at least one malicious link drops more than α percentage of packets (where α is the per-link drop rate threshold). Thus in the worst case (with the most natural packet drop loss, i.e., using α as the per-link drop rate), by Lemma 1 we can compute ψ_{th} as: $\psi_{th} = 1-(1-\alpha)^{2d}$.

If each link l_i has a drop rate $\theta_i < \alpha$, the end-to-end drop rate ψ_d is given by:

$$\psi_d = 1 - [\prod_{i=0}^{d-1}(1-\theta_i)]^2 < 1-(1-\alpha)^{2d} \Rightarrow \psi_d < \psi_{th}$$

Thus when $\psi_d > \psi_{th}$, there must be at least one malicious link. Then S derives the individual drop rate θ_i of each link l_i by using the results from Lemma 2. By comparing each θ_i with α, S can identify the malicious links. Note that in the converged condition, there is no false negative, while the false positive is given by Theorem 4.

Now we compute the maximum end-to-end drop rate that an adversary can cause without being detected, i.e., without causing $\psi_d > \psi_{th}$. Suppose there are z malicious links with the drop rate $\theta_{\mathcal{M}_1}, \ldots, \theta_{\mathcal{M}_z} > \alpha$. Given the fixed threshold ψ_{th}, when the malicious links can cause maximum drop rate with $\psi_d < \psi_{th}$, we have:

$$1-(1-\rho)^{2(d-z)} \cdot [\prod_{k=1}^{z}(1-\theta_{\mathcal{M}_k})]^2 = 1-(1-\alpha)^{2d}$$

Therefore, z malicious links can drop at most

$$1-[\prod_{k=1}^{z}(1-\theta_{\mathcal{M}_k})]^2 = 1-\frac{(1-\alpha)^{2d}}{(1-\rho)^{2(d-z)}}$$

percentage of traffic without being detected. ■

Proof of Theorem 2: Leveraging $(1-x)^n = 1-nx$ when $x \to 0$, we can transform the formula of ζ in Theorem 1 as:

$$\begin{aligned}
\zeta =& 1-\frac{(1-\alpha)^{2d}}{(1-\rho)^{2(d-z)}} \doteq 1-\frac{1-2d\cdot\alpha}{1-2(d-z)\cdot\rho} \\
=& \big(2d(\alpha-\rho)+2z\rho\big)\big(1-2(d-z)\rho\big)^{-1} \doteq \big(2d(\alpha-\rho)+2z\rho\big)\big(1+2(d-z)\rho\big) \\
=& 2d(\alpha-\rho)+\rho\cdot\Big(4d^2(\alpha-\rho)+z\cdot\big(2-4d(\alpha-\rho)\big)\Big)+\rho^2\cdot 2z\cdot 2(d-z)
\end{aligned}$$

We can further suppose $\epsilon = \alpha - \rho$ is constant, and neglect the second order ρ^2 term, then the above formula transforms to:

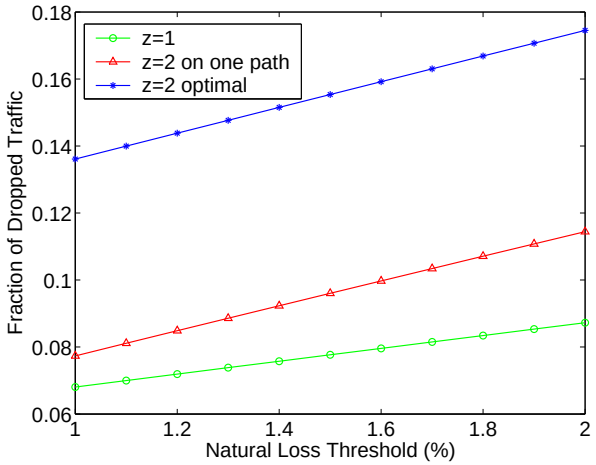
$$\zeta = 2d\epsilon + \rho\cdot\big(4d^2\epsilon + z(2-4d\epsilon)\big)$$

This proves that ζ increases proportionally to ρ. Next we show that the optimal strategy of the adversary (to drop the maximum traffic with z compromised links) is to deploy only one malicious link for one path.

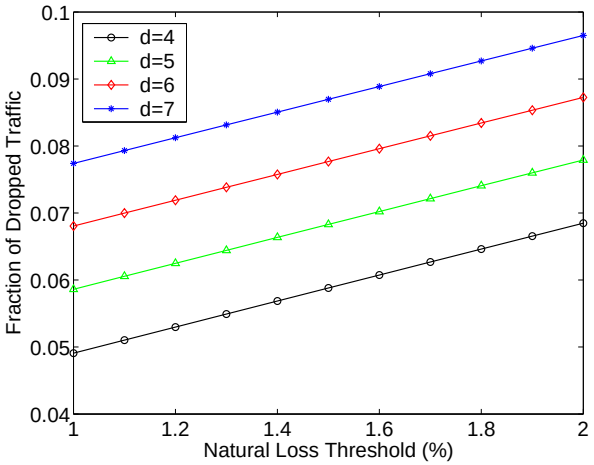
We give a sketch of the proof by showing two extreme cases to present the intuition.

1. In one extreme case where the adversary deploys all z compromised links on one path, the adversary can drop at most:
$$\zeta_1 = 2d\epsilon + \rho\cdot\big(4d^2\epsilon + z(2-4d\epsilon)\big)$$
traffic as calculated above.

2. In the other extreme case where the adversary deploys one compromised link for one path (thus z paths contain a comprised link), the adversary can drop at most:
$$\zeta_2 = z\Big(2d\epsilon + \rho\cdot\big(4d^2\epsilon + 1\cdot(2-4d\epsilon)\big)\Big)$$
traffic.



(a) Fraction of dropped traffic with different natural drop rate thresholds (α) and numbers of attackers, given $d = 6$



(b) Fraction of dropped traffic at different paths with varied lengths (d) and natural drop rate thresholds (α).

Figure 3: The fraction of traffic the adversary can drop under various settings.

Obviously we have $\zeta_1 \leq \zeta_2 \leq z \cdot \zeta_1$; and in Case 2 ζ increases linearly with the increase of z. ■

Proof of Theorem 3: Suppose that a malicious link $l_{\mathcal{M}_k}$ ($k = 1, 2, \ldots, z$) drops data, probe and ack packets at different rates, denoted by $\mu_{\mathcal{M}_k}$, $\nu_{\mathcal{M}_k}$ and $\omega_{\mathcal{M}_k}$ respectively. Given the fixed threshold ψ_{th}, the malicious links can cause maximum drop rate when $\psi_d < \psi_{th}$, yielding:

$$1 - (1-\rho)^{2(d-z)} \cdot \prod_{i=1}^{z}(1-\mu_{\mathcal{M}_i})(1-\nu_{\mathcal{M}_i})(1-\omega_{\mathcal{M}_i}) < \psi_{th}$$

Therefore, the adversary can drop at most

$$1 - \prod_{i=1}^{z}(1-\mu_{\mathcal{M}_i})(1-\nu_{\mathcal{M}_i})(1-\omega_{\mathcal{M}_i}) = 1 - \frac{(1-\alpha)^{2d}}{(1-\rho)^{2(d-z)}}$$

percentage of packets without being detected. This yields the same result as Theorem 1. ■

Proof of Theorem 4: In the following proof, we first study how many packet transmissions are required to estimate the drop rate of a single link l_i within a certain *accuracy interval*: suppose that the true value of drop rate of l_i is θ_i^*, and the estimated drop rate of l_i is θ_i, we compute the number of packets needed to achieve a $(\epsilon_{\theta_i}, \sigma)$-accuracy for θ_i:

$$Pr(|\theta_i - \theta_i^*| > \epsilon_{\theta_i}) < \sigma$$

i.e., with probability $1 - \sigma$ the estimated θ_i is within $(\theta_i^* - \epsilon_\theta, \theta_i^* + \epsilon_\theta)$. Then we compute the total number of packets needed to achieve a $(\epsilon_{\theta_i}, \sigma)$-accuracy for *every* link's θ_i. In the following we analyze the full-ack scheme and PAI in turn.

I. Full-ack Scheme. We first study a given link l_i. Let θ_i be the estimated drop rate of link l_i, and p_i be the observed *conditional* probability that link l_i correctly forwards both a data packet and the returning ack *given that* a data packet reaches the upstream end of l_i, i.e., node f_i in Figure 1. Thus we have:

$$p_i = (1-\theta_i)^2 \tag{2}$$

We define each time a data packet reaches node f_i as a random trial of l_i (or trial for l_i in short), and define the indicator random variable $X_j \in \{0, 1\}$ such that $X_j = 0$ if link l_i results in an acknowledgement failure for the jth trial; otherwise $X_j = 1$. Denote by p_i^* the true value of p_i, then we have $X_j \sim Bernoulli(p_i^*)$. In the full-ack scheme, the source can learn the numbers of trials for l_i where $X_j = 0$ and $X_j = 1$, denoted by $count(X_j = 0)$ and $count(X_j = 1)$ respectively. Therefore the source can estimate:

$$p_i = \frac{count(X_j = 1)}{count(X_j = 0) + count(X_j = 1)}$$

as the *Maximum Likelihood Estimation* of p_i^*. Given the number of trials N_i for l_i, by using *Hoeffding's inequality*, we have:

$$Pr(|p_i - p_i^*| > \epsilon_{p_i}) < 2e^{-2N_i\epsilon_{p_i}^2} \quad \Rightarrow \quad N_i = \frac{ln(\frac{2}{\sigma})}{2\epsilon_{p_i}^2} \tag{3}$$

Note that we are not interested in ϵ_{p_i}, but ϵ_{θ_i} instead. However θ_i cannot be directly estimated, but must be derived from Equation 2, i.e.:

$$\theta_i = 1 - p_i^{\frac{1}{2}} \tag{4}$$

Given the error ϵ_{p_i} of p_i, we can derive the error ϵ_{θ_i} of θ_i from Equation 4 using the *Uncertainty Propagation Rule*:

$$\epsilon_{\theta_i} = |\frac{\partial \theta_i}{\partial p_i}\epsilon_{p_i}| = \frac{1}{2}p_i^{-\frac{1}{2}} \cdot \epsilon_{p_i} \quad \Rightarrow \quad \epsilon_{p_i} = 2\epsilon_{\theta_i} \cdot p_i^{\frac{1}{2}} \tag{5}$$

Combining Equations 3, 2 and 5, and given $\epsilon_{\theta_i} \leq \epsilon$ we have:

$$N_i = \frac{ln(\frac{2}{\sigma})}{2\big(2\epsilon_{\theta_i} \cdot p_i^{\frac{1}{2}}\big)} = \frac{ln(\frac{2}{\sigma})}{8\epsilon_{\theta_i}^2 \cdot (1-\rho)^2} \geq \frac{ln(\frac{2}{\sigma})}{8\epsilon^2 \cdot (1-\rho)^2} \tag{6}$$

Now we compute the number of packets needed to give an estimate with (ϵ, σ)-accuracy for every link in a given path. When each packet transmitted by the source can reach node f_{d-1}, it provides a trial for every link l_i. Therefore transmitting N_i packets to f_{d-1} also suffices to give other links enough trials, which requires

$$N_{d-1}\frac{1}{(1-\alpha)^d} = \frac{ln(\frac{2}{\sigma})}{8\epsilon^2 \cdot (1-\rho)^{2+d}}$$

total number of packets transmitted from the source.

II. PAI. Let θ_k be the estimate drop rate of link l_k. Let the event that node f_{k+1} is *selected* be a random trial for l_k (or trial for l_k for short). Let p_k be the observed *conditional* probability that node f_{k+1} fails to ack when f_{k+1} is *selected*, and p_k^* be the true value of p_k. We define the indicator random variable $Y_j \in \{0, 1\}$ such that $Y_j = 0$ if node f_{k+1} fails to ack when f_{k+1} is *selected* the jth time; otherwise $Y_j = 1$. Then we have $Y_j \sim Bernoulli(p_k^*)$. Therefore the source can directly estimate p_k as

$$p_k = \frac{count(Y_j = 0)}{count(Y_j = 0) + count(Y_j = 1)}$$

which is the Maximum Likelihood Estimation of p_k^*. Given the number of trials N_k for l_k, by using Hoeffding's inequality, we have:

$$Pr(|p_k - p_k^*| > \epsilon_{p_k}) < 2e^{-2N_k\epsilon_{p_k}^2} \quad \Rightarrow \quad N_k = \frac{ln(\frac{2}{\sigma})}{2\epsilon_{p_k}^2} \tag{7}$$

Again, since we are not interested in ϵ_{p_k}, but ϵ_{θ_k} instead, we derive ϵ_{θ_k} from ϵ_{p_k} in the following.

Using our notations where $count(Y_j = 0) = \delta_k$ and $count(Y_j = 0) + count(Y_j = 1) = \eta_{k+1}$, we have $p_k = \frac{\delta_k}{\eta_{k+1}}$. Leveraging Lemma 1, we further have:

$$p_k = \begin{cases} 1 - (1-\theta_0)^3, & k = 0 \\ 1 - [\prod_{y=0}^{k}(1-\theta_y)]^3 = 1 - C \cdot (1-\theta_k)^3, & \text{where } C = \prod_{y=0}^{k-1}(1-\theta_y)^3, \quad k \geq 1 \end{cases} \tag{8}$$

Then we can infer θ_k from p_k from Equation 8 as:

$$\theta_k = \begin{cases} 1 - (1-p_0)^{\frac{1}{3}}, & k = 0 \\ 1 - [\frac{1}{C}(1-p_k)]^{\frac{1}{3}}, & \text{where } C = \prod_{y=0}^{k-1}(1-\theta_y)^3, \quad k \geq 1 \end{cases} \tag{9}$$

Using the *Uncertainty Propagation Rule* based on Equation 9 we have:

1. For k = 0,

$$\epsilon_{\theta_0} = |\frac{\partial \theta_0}{\partial p_0} \cdot \epsilon_{p_0}| = \frac{1}{3}(1-p_0)^{-\frac{2}{3}} \cdot \epsilon_{p_0} \tag{10}$$

2. For $k \geq 1$,

$$\epsilon_{\theta_k}^2 = \sum_{y=0}^{k-1} \left(\frac{\partial \theta_k}{\partial \theta_y} \cdot \epsilon_{\theta_y}\right)^2 + \left(\frac{\partial \theta_k}{\partial p_k} \cdot \epsilon_{p_k}\right)^2 \tag{11}$$

In the following we solve Equation 11 for $k \geq 1$. Leveraging $1 - p_k = C(1 - \theta_k)^3$ from Equation 8, and approximating $\theta_y = \rho, y = 1, \ldots, k$, by solving the partial derivatives $\frac{\partial \theta_k}{\partial \theta_y}$ and $\frac{\partial \theta_k}{\partial p_k}$ we have:

$$\begin{aligned} \frac{\partial \theta_k}{\partial \theta_y} &= C^{-\frac{1}{3}}(1 - p_k)^{\frac{1}{3}}(1 - \theta_y)^{-1} = 1, \\ \frac{\partial \theta_k}{\partial \theta_y} &= \frac{1}{3}C^{-\frac{1}{3}}(1 - p_k)^{-\frac{2}{3}} = \frac{1}{3}C^{-1}(1 - \rho)^{-2} \end{aligned} \tag{12}$$

Combining Equations 11 and 12 yields:

$$\epsilon_{\theta_k}^2 = \sum_{y=0}^{k-1} \epsilon_{\theta_y}^2 + \frac{1}{9}C^{-2}(1 - \rho)^{-4}\epsilon_{p_k}^2 \tag{13}$$

Note that Equation 14 is recursive on $\epsilon_{\theta_k}^2$, solving this recurrence yields (given $\epsilon_{\theta_k} \leq \epsilon$):

$$\epsilon_{\theta_k}^2 \leq \epsilon_{\theta_0}^2 \doteq \frac{\epsilon_{\theta_k}^2}{2^k} \leq \frac{\epsilon^2}{2^d} \tag{14}$$

Combining Equations 10 and 7, we have:

$$N = 2^d \frac{ln(\frac{2}{\sigma})}{18\epsilon^2(1 - \rho)^{\frac{4}{3}}} \tag{15}$$

Now we compute the total number of packets needed to give a (ϵ, σ)-accuracy estimate for every link' drop rate in a given path. We abstract a random trial for link l_k as *coupon* k, then a path with length d has d different coupons. The problem is to compute the expected wait time (number of trials) to gather N copies for each coupon k. When $N = 1$ the problem reduces to the classic *Coupon Collector* problem, which has an expected wait time $O(d \cdot log(d))$. With $N \neq 1$, the wait time has a simple upper bound:

$$O(N \cdot d \cdot log(d)) = O(2^d \frac{ln(\frac{2}{\sigma})}{18\epsilon^2(1 - \rho)^{\frac{4}{3}}} \cdot d \cdot log(d))$$

This proves the theorem. ■

B Overhead Analysis of PAI

Communication. Here we analyze the communication overhead incurred by the PAI protocol. Recall from the description of PAI that each intermediate node f_i on the forwarding path either (a) generates a new ack if no ack is received from downstream or when f_i is selected, or (b) re-encrypts the ack received from downstream . Therefore, an ack packet traversing the path has a constant size ($\Theta(1)$) at any point of time. Further, the constant-size probe packet also contributes to the communication overhead. Therefore, the overall communication overhead for PAI is $O(1)$ per data packet sent by the source.

Storage. On receiving a data packet m, an intermediate node f_i must maintain a storage state for m until it either receives an ACK from downstream or its wait timer expires. Now we calculate the buffer size required to maintain the storage state for the data packets.

Suppose the maximum transmission rate of the source is ν packets per unit time; and recall that r_i denotes the round trip time from node f_i to the base station. When a node f_i receives a packet m, the following cases are possible:

1. f_i receives an ack from the base station. In this case, it deletes the state maintained for m. Thus, overall f_i must maintain storage state for m for time r_i. Therefore f_i needs to buffer $\nu \cdot r_i$ packets in this case .

2. f_i receives no ack from the base station within wait-time r_i. In this case, f_i must further wait for the probe packet from the source (this amounts to $r_0 - r_i$ time) and then wait for the downstream ack (this amounts to r_i time). Therefore, S must maintain storage state for m for an additional time $(r_0 - r_i) + r_i$ since the expiration of the wait-timer. Thus in this case f_i needs to buffer $\nu \cdot (r_0 + r_i)$ packets.

Suppose among all the packets transmitted from the source, Cases 1 and 2 account for $1 - p$ and p out of them respectively, then the storage overhead for an intermediate node f_i is given by:

$$\nu \cdot [(1-p) \cdot r_i + p \cdot (r_0 + r_i)] \leq \nu \cdot [(1-p) \cdot r_0 + p \cdot 2 \cdot r_0] \leq \nu \cdot (1+p) \cdot r_0$$

Note that p must be bounded by the fraction $\psi_{th}$of packets the adversary can drop, i.e., $p \leq \psi_{th}$, where ψ_{th} is the end-to-end drop rate threshold given by Theorem 1.

C Rate Limiting Protocol: Proofs

Definition 7 A **nonincreasing outflow completion** of a view $E(S)$ is a communication pattern C' such that, for any directed edge $e = (c \rightarrow p) \in E(S)$ with weight w_e in $E(S)$, if e is an outgoing edge from S (i.e., $c \in S$ and $p \notin S$) then C' assigns at most weight w_e to e; otherwise if e is not an incoming or internal edge of S then C' assigns the same weight w_e to e.

Proof of Lemma 3: When i and j verify the hash of a against the topological hash DAG construction, they both verify that the entire downstream topology of a is reconstructed correctly. Specifically, the full set of paths from a to the base station must be represented correctly in the topological hash DAG. Since i and j have a common (correct) view of the paths from a to the base station, this uniquely determines the position of a in the hash DAG; since computing hash collisions is computationally infeasible for the attacker, it is overwhelmingly likely that i and j must thus have received the same report from a. ■

Proof of Lemma 4: By induction on the size of $S(U)$. Base case: a single node v_1 with no descendants. It has no internal edges, so we are done. Now consider any U. Since the network is a DAG, there must exist some node in S which only has parents which are not in S'. WLOG let that node be v_k. Let the children of v_k be $u_1, \ldots, u_q$; these child nodes must all be in $S(U)$. Since the edge capacities were assigned conservatively, the weights on the outgoing edges of v_k is no more than the origination rate limit of v_k plus the total capacity of the incoming edges to v_k. Hence, we can arbitrarily assign weights to the incoming edges from $u_1, \ldots, u_q$ in such a way that v_k does not exceed its allotted origination rate limit. Consider $U' = U - v_k + u_1 + \cdots + u_q$. Clearly, $S(U') = S(U) - v_k$. Hence the induction hypothesis holds and we can assign weights to the internal edges of $S(U')$ such that the weights respect edge capacities and no node in $S(U')$ exceeds its allotted origination rate limit. Hence the theorem holds for $S(U)$. ■

Theorem 7 *General restatement of Theorem 5: If there exists a nonincreasing outflow completion C' of a legitimate view $E(S)$ that is well-behaved, then there exists a completion C of $E(S)$ that is well-behaved.*

Proof of Theorem 7: Consider a communication pattern C which assigns the same weight as the legitimate view L on the outgoing edges of L (i.e. the set E') but the same weight as C' everywhere else. If a malicious node obeys its rate limit in C', it could only have its total incoming traffic increased in C, so it obeys its rate limit in C. Since every legitimate node obeys its rate limit in L, and C is a completion of L, we have that C is well-behaved, hence L is permissible. ■

Proof of Theorem 6: Let the set of all legitimate nodes be L. Let $A(L)$ be the set which includes L and all ancestors of the nodes in L. By Lemma 3, every node in $A(L)$ reports the same traffic report to all its legitimate descendants. Furthermore, the traffic reports of the nodes in $A(L)$ are all mutually consistent (i.e. no downstream node claims a higher amount of traffic over a link than its upstream neighbor). Also, since each legitimate node passed its distributed verification phase, every traffic report of every node in $A(L)$ obeys the rate limit for that node. Hence, to construct a well-behaved communication pattern P that is a completion of the legitimate view, we simply assign weights to the internal and incident edges of $A(L)$ based on the traffic reports of the nodes in $A(L)$: for every edge $e = (u, v)$ where either $u \in A(L)$ or $v \in A(L)$, and e is not part of the legitimate view, if only one of u, v is in $A(L)$, we assign to e the weight of e in the traffic report of the node in $A(L)$. If both $u, v \in A(L)$,

then we assign to e the minimum of the two traffic weights assigned to e from the traffic reports of u and v. Once this is done, it remains to show that we can assign traffic weights to the remaining edges not internal to or incident to $A(L)$. Let V represent the set of all nodes in the network; note that $V - A(L)$ is the set of all malicious nodes which do not have a legitimate node as a child. Hence we can apply Lemma 4 to the set $V - A(L)$ to show that there exists a weight assignment for the remaining edges such that the overall communication C' is well-behaved. The weights assigned by C' are consistent with the legitimate view except possibly for outgoing edges from L where C' may assign a lower traffic weight. However by the same reasoning as the proof of Theorem 5, since C' is well-behaved, there must exist a well-behaved communication pattern C that is a completion of the view of L. ■