Packet-dropping Adversary Identification for Data Plane Security*

Xin Zhang
Carnegie Mellon University
xzhang1@cmu.edu

Abhishek Jain
UCLA
abhishek@cs.ucla.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

ABSTRACT

Until recently, the design of packet dropping adversary identification protocols that are robust to *both* benign packet loss *and* malicious behavior has proven to be surprisingly elusive. In this paper, we propose a *secure* and *practical* packet-dropping adversary localization scheme that is robust and achieves a high detection rate and low communication and storage overhead – the three key performance metrics for such protocols in realistic settings. Other recent work just optimizes *either* the detection rate *or* the communication overhead.

In this paper, we *systematically* explore the design space of acknowledgment-based protocols to identify a packet dropping adversary on a forwarding path. In particular, we investigate a set of basic protocols, each exemplifying a design dimension, and examine the underlying tradeoff between the performance metrics. For each basic protocol, we present both upper and lower performance bounds via theoretical analysis, and average-case results via simulations. We conclude that the proposed PAAI-1 protocol outperforms other related schemes.

1. INTRODUCTION

Even given a secure routing infrastructure, a malicious forwarding node can drop packets to reduce legitimate network throughput. Thus, packet-dropping attacks constitute a major threat to data plane security.

*This research was supported in part by CyLab at Carnegie Mellon under grants DAAD19-02-1-0389 and MURI W 911 NF 0710287 from the Army Research Office, grants CNS-0347807, CT-CS 0433540 and CNS-0627357 from the National Science Foundation. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, CMU, NSF, or the U.S. Government or any of its agencies.

ACM CoNEXT 2008, December 9-12, 2008, Madrid, SPAIN

A common protocol building block to identify a packet-dropping adversary located on a forwarding path is for the source node to require acknowledgement packets (ack) from the destination and the intermediate nodes. In a realistic setting, the forwarding links may incur some benign packet loss due to congestion and/or channel errors. At the same time, an adversary who potentially controls multiple intermediate nodes may try to bias the identification procedure by selectively dropping (and altering or injecting) packets in order to evade detection or incriminate honest nodes. Consequently, a secure *ack-based adversary identification* (AAI) protocol must be simultaneously robust to both benign packet loss and malicious behavior. In other words, it must exhibit low false positive (falsely identifying a legitimate link as malicious) and false negative (falsely leaving a malicious link undetected) rates.

However, until recently, the design of such protocols has proven to be surprisingly difficult, as exemplified by several insecure protocols in the literature [1, 2, 6, 11, 12, 15, 16] (see §2). Furthermore, such an AAI scheme must also be *practical*; in particular, it must possess *all* of the following properties: (a) fast detection rate (i.e., the time required to accurately localize the adversary), (b) low communication overhead, and (c) low storage overhead. Failing to achieve any of the above three properties may render the protocol impractical. For example, an AAI protocol with high communication and/or storage overhead will perform poorly even when the forwarding path is not under attack; this will be unacceptable in most settings, and especially in resource-constrained networks such as sensor networks. Similarly, an AAI protocol with a low detection rate will enforce only a poor bound on an adversary's ability to degrade end-to-end throughput before being identified. This may result in a significant monetary loss to a service provider and, worse, in cases where routing paths change periodically, the attacker may escape unscathed. Unfortunately, prior work optimizes *either* the detection rate [3, 4], *or* the communication overhead [7]. Therefore, AAI protocols that achieve a good trade-off between the three aforementioned performance metrics are more desirable than those that optimize only a particular metric.

In this paper, we systematically explore the design space of AAI protocols with the goal of designing a practical (and secure) AAI protocol. In particular, we propose a set of basic

protocols, where each protocol exemplifies a distinct design dimension. Such an approach helps us examine the underlying tradeoff between the different performance metrics, and lets us obtain a *practical* and secure AAI protocol even in the presence of multiple adversarial packet-dropping nodes.

We observe that designing *any* AAI protocol involves making two fundamental decisions:

1. which data packets to acknowledge; and
2. which intermediate nodes should respond to the ack request.

With this in mind, we explore different design choices along the two aforementioned aspects and investigate the tradeoff between the performance metrics. More specifically, we study the following approaches:

1. A strawman approach: *Every* intermediate node sends an ack for *every* lost data packet.
2. The Probabilistic Ack-based Adversary Identification (PAAI) approaches: either (a) only a subset of data packets must be acknowledged (PAAI-1), or (b) only a subset of intermediate nodes must respond to an ack request (PAAI-2).

The full-ack scheme achieves the best detection rate by determining the link for *every single* packet drop. However, gathering such *fine-grained* information introduces high communication overhead. In contrast, PAAI-1 and PAAI-2 employ probabilistic sampling to gather only *coarse-grained* information, differing from each other in that they perform probabilistic sampling in different dimensions. In both PAAI schemes, we aim to achieve a high detection rate while retaining practicality for most networks.

The PAAI-1 protocol is fairly intuitive, simple and flexible, yet achieves more desirable properties than the full-ack scheme, PAAI-2, and other related work. Finally, we also discuss the viability of constructing protocols that exemplify hybrids of the basic design primitives (§10).

Contribution. To the best of our knowledge, this is the first attempt to design a secure AAI protocol that obtains a practical trade-off between the detection rate and the communication and storage overhead for realistic network settings. It is also the first systematic study of the design space for AAI protocols (§4, §5 and §6). We propose a set of basic AAI protocols, one for each design dimension, where the PAAI-1 protocol (§6.1) is distinctly more practical than the others. We obtain theoretical bounds for the performance of our protocols (§7), and also launch simulations to derive average-case results and validate our theoretical results (§8).

2. RELATED WORK

A related line of research aims to only detect malicious packet dropping activity [5, 9, 13], while in this paper, we are interested in the more difficult problem of *localizing* the packet dropping adversary. We now discuss prior work in the localization category.

Since the work of Perlman [16], several works leveraged acknowledgment-based approaches to identify Byzantine adversaries at the network layer. Unfortunately, most of these protocols either incur high overhead, or fail to achieve security in the presence of colluding adversaries and natural packet loss (i.e., they fail to provide any effective bound on an adversary's ability to degrade network throughput before being identified). Some vital vulnerabilities of certain related work [1,6,15] have been summarized by Barak et al. [7]. The protocols designed by Liu et al. [12] and Mike et al. [11] fail to prevent colluding nodes from incriminating honest links as malicious. Argyraki et al. propose a counting-based protocol [2] in a restricted adversarial model where an attacker can only drop but not alter or inject packets; consequently, the protocol is insecure in our strong adversary model (see §3.2). Avrampoulos et al. [3, 4] present a secure routing protocol by combining techniques such as source routing, hop-by-hop authentication and ack-based probing. However, for a *forwarding path* of length d, their scheme induces a key storage overhead of $O(d^2)$ and a high communication overhead equivalent to that of the strawman approach described later (§4,§7,§8).

Recently, Barak et al. [7] proposed a set of fault localization (FL) protocols for the Internet. However, we note that their statistical FL protocol is mainly optimized to reduce communication overhead; and consequently achieves a rather poor *best case* detection rate on the order of 10^6 packets. In contrast, our PAAI-1 protocol offers a flexible trade-off between the performance metrics. In particular, although PAAI-1 incurs more overhead (but still low enough for a practical scenario) than the statistical FL protocol, it achieves a detection rate that is almost *three* orders of magnitude faster. For example, assuming the source's sending rate of 100 packets per second, PAAI-1 converges in 8 minutes, while the statistical FL protocol requires 50 hours for similar parameter settings! We stress that, in most network settings, the trade-off offered by the PAAI-1 protocol is more desirable than the combination of very low overhead and poor detection rate offered by the statistical FL protocol (see §7 and §8 for details).

In a related line of research [8, 10, 14] that focuses on a distributed monitoring approach to detect malicious routers by using a traffic validation mechanism based on the law of conservation of flow. However, that line of research requires considerable overhead in terms of storing and communicating aggregated yet fine-grained per-flow state (e.g., fingerprints of packets, packet ordering etc.) [14].

3. FORMAL PROBLEM SETTING

In this section, we first present the problem definition and metrics we use to evaluate the performance of the protocols. Next, we discuss our assumptions and basic notation.

3.1 Problem Definition

Given a set of adversarial nodes located on a forwarding path between a source and destination node, we are interested in the design of protocols that enable the *source*

to monitor the forwarding path for packet dropping activity over a period of time and then securely localize the presence of the adversary on a particular link (or a set of links). We build on the approach of requiring acknowledgment packets from the destination and the intermediate nodes on the forwarding path. We refer to such protocols as *ack-based adversary identification* (AAI) protocols. The literature shows that such protocols can only identify links adjacent to malicious nodes, rather than identifying the nodes [7]. We consider realistic network settings and a strong adversary model (see §3.2 below).

Performance Metrics. In this paper, we consider AAI protocols that utilize symmetric key cryptography.[1] We identify three key metrics to evaluate such protocols, (a) *detection rate*, i.e., the number of data packet transmissions required to detect a malicious link (with the false positive and negative rates below a certain threshold), (b) *communication overhead*, i.e., the additional packets (and their size) that are sent per data packet from the source, and (c) *storage overhead*, i.e., the amount of temporary storage that must be maintained at each intermediate node per unit time.

Given the above definitions, one would seek an AAI protocol that (a) achieves detection time (defined as the detection rate divided by the source's packet sending rate) in the order of minutes or less, (b) incurs communication overhead of less than 10% of the normal traffic, and (c) requires storage overhead in the order of tens of kilobytes or less. Looking ahead, the proposed PAAI-1 protocol achieves the aforementioned trade-off between the performance metrics, while other related protocols fail to achieve practical values for at least one metric.

3.2 Assumptions

Network Assumptions. We assume a general multi-hop network, and the presence of a routing infrastructure with a certain lower bound on the time before the routing paths change in the network.[2] We assume that the links in the network independently exhibit some natural packet loss due to congestion and/or channel errors. In this paper, we only consider *unicast* messages. Following the literature, we will focus on a given forwarding path between a source and a destination pair. Further, we assume the presence of *symmetric* paths, where the forward path (for data) and reverse paths (for acknowledgements) are identical; and we assume that a source node knows its forwarding path to the destination. Finally, we assume that the nodes on any given path are loosely time-synchronized.

Security Infrastructure. Given a path from a source to a destination, we assume that the source shares a pairwise symmetric key with each intermediate node on the path to the destination. We further assume that the nodes can perform symmetric key operations as well as compute a collision-resistant hash function h and a keyed pseudorandom function PRF.

[1] Although a fairly simple AAI protocol that employs asymmetric key cryptography exists (described in the full version due to lack of space [17]), we note that protocols employing asymmetric key cryptography are generally undesirable due to their high per-packet computation and communication overhead.

[2] This is necessary since any AAI protocol requires some time to converge in the face of natural packet loss.

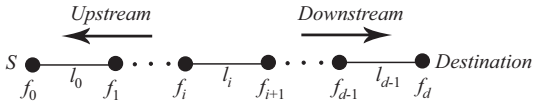


Figure 1: Example topology

Adversary Model. We assume that the source and the destination nodes on the path being monitored are honest. We assume an adversary with polynomially bounded computational power is in complete control of an arbitrary number of intermediate nodes on the path, including knowledge of their secret keys. The adversary can eavesdrop and perform traffic analysis anywhere on the path. The adversary may drop, inject or alter packets on the links that are under its control. We allow the protocol parameters to be public; consequently, the adversary may try to bias the measurement results in order to evade detection or incriminate honest links. Finally, we assume that the adversary cannot influence the natural packet loss rate on the links on the path.

3.3 Basic Notation

We consider one source $\mathbb{S}$ and a path of length d hops to a destination $\mathbb{D}$. The nodes along the path are denoted as $\mathbb{F}_0, \ldots, \mathbb{F}_d$, where $\mathbb{F}_i$ is i hops away from $\mathbb{S}$ (see Figure 1, where $\mathbb{F}_0 = \mathbb{S}$). We call nodes closer to $\mathbb{D}$ *downstream* nodes, and nodes that are further away from $\mathbb{D}$ as *upstream* nodes. Let l_i be the link between node pair $\mathbb{F}_i, \mathbb{F}_{i+1}$. We denote the round-trip time from a node $\mathbb{F}_i$ to $\mathbb{F}_d$ as r_i. $\mathbb{S}$ shares a pairwise symmetric key K_i with each intermediate node $\mathbb{F}_i$. Let $\mathrm{E}_K(\cdot)$ denote encryption using symmetric key K. Further, let $[m]_K$ denote a message m authenticated by key K using a message authentication code (MAC). 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.

Ack Structure. For any data packet m sent out by $\mathbb{S}$, let the hash of m, denoted by $\mathrm{H}(m)$, be a packet identifier for m. For any m, we define the corresponding ack from $\mathbb{F}_i$ to have the structure $a_i = \langle \mathrm{H}(m) || \mathcal{A}_i^m \rangle$, where $\mathcal{A}_i^m$ is a report computed by $\mathbb{F}_i$. The report $\mathcal{A}_i^m$ will be a function of $\mathbb{F}_i$'s *local* report $\mathcal{R}_i$ and its downstream neighbor $\mathbb{F}_{i+1}$'s ack (if present). Specific details may vary in each protocol description.

Onion Reports. We recall the well-known notion of an *onion report*. When each intermediate node $\mathbb{F}_i$ must return a local report $\mathcal{R}_i$ to $\mathbb{S}$ in an authenticated manner, then we have inductively, for $i \in [1, d-1]$, $\mathcal{A}_i = [i || \mathcal{R}_i || \mathcal{A}_{i+1}]_{K_i}$, while $\mathcal{A}_d = [d || \mathcal{R}_d]_{K_d}$.

4. A STRAWMAN APPROACH: FULL-ACK

We observe that designing *any* AAI protocol involves making two fundamental decisions: (a) which data packets to acknowledge, and (b) which intermediate nodes should respond to the ack request. As a first step towards a systematic exploration of the protocol design space for AAI protocols along the two aforementioned aspects, we discuss the simple and fairly intuitive 'full-ack' protocol (similar to the Optimistic Per-Packet FL Protocol from Barak et al. [7]), where *every* intermediate node on the forwarding path must return an ack for *every* lost[3] data packet sent by the source. Below, we give a brief description of the protocol and discuss its security and performance. A theoretical analysis and simulation results for the full-ack protocol are given later in §7 and §8 respectively.

Protocol. Let us consider that $\mathbb{S}$ sends out a data packet m towards the destination $\mathbb{D}$. On receiving m, $\mathbb{D}$ must return an ack, a_d, authenticated with the secret key shared with $\mathbb{S}$, i.e., $a_d = [\mathrm{H}(m)]_{K_d}$. If no valid ack is received from $\mathbb{D}$ within a pre-specified wait-time, $\mathbb{S}$ will send out an onion report request. The onion report is computed by the intermediate nodes in the manner explained earlier, wherein a local report $\mathcal{R}_i$ is set to be $\langle i||\mathrm{H}(m)||a_d\rangle$. Upon receiving the ack containing the onion report from $\mathbb{F}_1$, $\mathbb{S}$ can sequentially verify each report embedded in it. For some $i < d$, if the MAC from each intermediate node $\mathbb{F}_j, j \in [1, i]$ is valid but the MAC from $\mathbb{F}_{i+1}$ is invalid or not present in the final ack, then $\mathbb{S}$ identifies link l_i as faulty and adds one to its *drop score*. Over a period of time, if the drop score of a particular link exceeds a fixed threshold determined from the natural packet drop rate then that link is identified as malicious.

Security. In the above protocol, if a malicious node drops a packet (data or ack), one of its adjacent links has its drop score increased[4]. 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).

Performance. For *each* 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, this high detection rate is achieved at the price of a *large* amount of communication overhead at each node. Specifically, the full-ack scheme imposes an overhead of at least one packet of $\mathrm{O}(1)$-size per data packet sent out by $\mathbb{S}$; and an additional overhead of one packet of $\mathrm{O}(d)$-size (the onion report) in case of a drop. The storage overhead is high in the worst case but lower on average due to the fast detection rate. More details are given later in §7 and §8.

The high overhead makes the full-ack protocol unaffordable for most networks; therefore, AAI protocols with a better trade off amongst the three performance metrics are desirable.

[3]A lost packet signifies one that fails to reach the destination.

[4]This follows largely from the security of onion reports. Since PAAI-1 employs similar techniques, we defer more details to Section 5 in order to conserve space.

5. OVERVIEW OF PAAI

In contrast to the full-ack protocol, where the ack mechanism was completely deterministic, we now investigate probabilistic AAI (PAAI) approaches with the underlying motive of reducing the protocol overhead at the expense of slightly worsening the detection rate in order to achieve a more reasonable trade-off between the performance metrics. Loosely speaking, we investigate two contrasting approaches; one where only a subset of data packets must be acknowledged, and another where only a subset of intermediate nodes must ack[5]. In particular, we construct, (a) the PAAI-1 protocol: every intermediate node sends an ack for only a selected fraction of data packets; and (b) the PAAI-2 protocol: only one selected intermediate node sends an ack for each data packet. At first glance, the two approaches may seem to be only minor variations of the full-ack mechanism; however, we stress that there are several challenges involved in ensuring security of these approaches. We now briefly outline these approaches along with the challenges involved.

In the first approach (PAAI-1), $\mathbb{S}$ monitors the path for only a fraction of the total traffic. More specifically, for a given data packet, $\mathbb{S}$ solicits an ack from every intermediate node *only with some probability* p. Now, since a fraction of traffic is *unmonitored*, the protocol must ensure that a malicious node $\mathbb{F}_z$ is not able to determine from the content of a data packet m whether $\mathbb{S}$ solicits an ack for m. Otherwise, on receiving m, if $\mathbb{F}_z$ determines that m need not be acknowledged, then it could safely drop m without increasing its probability of being identified.

In PAAI-2, $\mathbb{S}$ monitors the path for every data packet, with the provision that $\mathbb{S}$ solicits an ack for a *lost* data packet from only one *selected* node on the path. However, the protocol must ensure that a malicious node $\mathbb{F}_z$ cannot decipher the identity of the selected node $\mathbb{F}_e$ from the content of a data packet m. Otherwise, on receiving m, if $\mathbb{F}_z$ determines that $\mathbb{F}_e \leq \mathbb{F}_z$ (i.e., whether $\mathbb{F}_e$ is upstream to or equal to $\mathbb{F}_z$), then it could safely drop m without increasing its probability of being identified.

In order to circumvent the above documented attacks and still perform probabilistic monitoring, we make use of a *delayed sampling* mechanism. Specifically, in both PAAI protocols, $\mathbb{S}$ sends out an ack request (henceforth referred to as a *probe*) at a later time for a data packet sent earlier. In PAAI-1, the probe conveys the information that the corresponding data packet must be acknowledged (otherwise no probe is sent). In PAAI-2, the probe content determines which intermediate node is selected. However, in either protocol, a malicious node may withhold a data packet until the arrival of the corresponding probe in an attempt to decide whether to drop m. To circumvent this, we require loose time-synchronization amongst the nodes in the network such

[5]It is natural to imagine the possibility of composing these approaches. We discuss this in Section 10.

that the clock error between two adjacent nodes $\mathbb{F}_i$ and $\mathbb{F}_{i+1}$ is less than $\min(r_0)$, i.e., the minimum value of the round trip time from $\mathbb{S}$ to the destination. In this scenario, an intermediate node would discard a data packet if it carries an expired timestamp.

Both PAAI protocols employ a scoring mechanism in order to identify malicious links over a period of time. We set a threshold for the end-to-end drop rate of data packets for a given path. The threshold value is chosen based on the natural drop rate, such that the natural end-to-end drop rate will not exceed the threshold value. At the end of each probe, $\mathbb{S}$ computes the end-to-end drop rate so far, based on the number of sent data packets and successfully received acks from the destination; if the drop rate exceeds the threshold value, then it indicates that an adversary is present on the path. Using the history of scores (i.e., the scores accumulated so far) of the links, $\mathbb{S}$ 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.

We now give some details on the specific scoring mechanism employed by each PAAI protocol. Loosely speaking, in PAAI-1, if an intermediate node fails to return an ack for a probed data packet, then $\mathbb{S}$ will increase the drop score of its upstream link. However, note that if each intermediate node were to send a separate ack, then a malicious node could selectively drop the acks from legitimate nodes in order to incriminate honest links. To circumvent this, PAAI-1 employs the use of onion reports similar to the full-ack protocol.

PAAI-2, on the other hand, utilizes a slightly different scoring mechanism. For a given data packet, if the selected node $\mathbb{F}_e$ fails to return an ack, then $\mathbb{S}$ infers that there exists at least one malicious link upstream of $\mathbb{F}_e$; consequently $\mathbb{S}$ will increase the drop score of *each* link between $\mathbb{F}_e$ and itself. Now, suppose that a malicious drop occurred at a link l_{i-1}. Then, let X be the event that the intermediate node $\mathbb{F}_i$ is selected. We ensure that event X occurs with a *fixed probability*. Due to the above scoring mechanism, each occurrence of X will create a difference in the scores of the links on either side of $\mathbb{F}_i$. Over a period of time, a difference in the score of two adjacent links would indicate a potential malicious link. In order to ensure that event X occurs with a fixed probability, PAAI-2 selects an intermediate node *uniformly at random* for any data packet. The protocol must also ensure that the identity of the selected node for any data packet is not revealed at *any point in time*; otherwise, a malicious node could selectively drop acks from legitimate nodes in order to incriminate honest links[6]. In order to circumvent this, we design an *oblivious* selection and acknowledgment procedure, such that the identity of the selected node is hidden to each node (except $\mathbb{S}$) *even through traffic analysis*.

Finally, we remark that an adversary may choose to alter or drop any of the following: (a) data packet, (b) probe, or (c) ack. However, our protocol design ensures that the source node $\mathbb{S}$ interprets each such activity simply as a data packet drop. In what follows, we will simply use the term *drop* to refer to any kind of packet alteration or drop. Looking ahead, in §7, we show that an adversary achieves the same total end-to-end drop rate by employing different individual drop rates for different packet types.

[6]Specifically, in order to incriminate an honest link l_h, a malicious node could drop the ack every time $\mathbb{F}_{h+1}$ is selected, while behaving honestly every time $\mathbb{F}_h$ is selected. This would create a difference between the scores of l_{h-1} and l_h.

6. THE PAAI PROTOCOLS

Formally, the two PAAI protocols, namely, PAAI-1 and PAAI-2 consist of five phases: (a) *send data and decide whether to probe*, (b) *probe*, (c) *acknowledge*, (d) *score*, and (e) *identify*. We give the details of both the protocols below.

6.1 PAAI-1

PAAI-1 employs probabilistic sampling in order to determine which data packets must be acknowledged. For every sampled data packet, PAAI-1 requires each intermediate node and the destination to return an onion report. The protocol details follow.

Phase 1: send data and decide whether to probe
Consider that $\mathbb{S}$ sends out a data packet $m = \langle \text{data}||\text{timestamp}\rangle$ towards the destination. On receiving m, an intermediate node $\mathbb{F}_i$ first checks whether the embedded timestamp is recent. If verification fails, then m is dropped. Otherwise, $\mathbb{F}_i$ stores the identifier $\mathrm{H}(m)$ for m and starts a wait timer $t_i = r_0/2$. Finally, m is forwarded toward the destination.

$\mathbb{S}$ then uses a secure sampling (SS) algorithm to determine whether it must send out a probe for m. When given any input m, the SS algorithm must output "Yes" with a fixed probability p, where p is the probe frequency fixed at setup time. Such an algorithm can easily be constructed by making use of a PRF keyed with a secret key known only to $\mathbb{S}$. Note that such a mechanism is necessary to prevent an adversary from correctly predicting whether or not a specific data packet is sampled.

If the SS algorithm outputs "No", then the protocol is terminated for the current round. Otherwise, $\mathbb{S}$ executes the next phase of the protocol. In the following, it is implicit that a node $\mathbb{F}_i$ accepts a packet (probe or ack) iff it contains a data packet identifier already stored at $\mathbb{F}_i$.

Phase 2: probe
$\mathbb{S}$ sends out a probe $c = \mathrm{H}(m)$ towards the destination. The probe contains the identifier $\mathrm{H}(m)$ for the data packet m sent earlier[7]. On receiving a probe, an intermediate node $\mathbb{F}_i$ starts a wait-timer $t_i = r_i$, forwards the probe towards the destination, and moves to the next phase.

Phase 3: acknowledge
In this phase, the destination $\mathbb{D}$ and intermediate nodes must

[7]Note that, in practice, the probe frequency p will be set to a very low value. Therefore, if we use unauthenticated probes, an adversary could potentially waste a lot of communication power of the intermediate nodes by sending bogus probes. As a countermeasure, one could use authenticated probe packets, where a chain of MACs (one for each intermediate node) is attached to each probe.

return an onion report to $\mathbb{S}$. Ideally, the onion report must either originate at the destination, or at the upstream node of the link where m_j was dropped. To this end, we employ the following rules: (a) If no downstream ack is received within the wait time t_i, $\mathbb{F}_i$ originates an onion report $\mathcal{A}_i = [i||\mathrm{H}(m)]_{K_i}$. (b) Otherwise, on receiving a downstream ack within the wait-time, $\mathbb{F}_i$ sets the local report $\mathcal{R}_i$ to be $\langle i||\mathrm{H}(m)\rangle$ to create an onion report $\mathcal{A}_i$ as explained earlier in §3.3. Finally, $\mathbb{F}_i$ sends out an ack $a_i = \langle \mathrm{H}(m)||\mathcal{A}_i\rangle$ towards $\mathbb{S}$.

Phase 4: score
Upon receiving the ack containing the onion report from $\mathbb{F}_1$, $\mathbb{S}$ can sequentially verify each report embedded in it. For some $i < d$, if the MAC from each intermediate node $\mathbb{F}_j, j \in [1, i]$ is valid but the MAC from $\mathbb{F}_{i+1}$ is invalid or not present in the final ack, then $\mathbb{S}$ identifies link l_i as faulty and adds one to its *drop score*[8].

Phase 5: identify
At any point in time, let s_i be the drop score of link l_i, and n be the total number of probes evoked by $\mathbb{S}$ so far. The average loss rate θ_i for link l_i so far can be computed as $\frac{s_i}{n}$. We set a *per-link drop rate threshold* (denoted by α) according to the natural loss rate ρ_i ($\alpha > \rho_i$). Then if $\theta_i > \alpha$, $\mathbb{S}$ convicts l_i as a malicious link. More details are given in §7.

6.2 PAAI-2

Now we turn to the other design alternative: probabilistically sampling a subset of intermediate nodes which must return an ack. We propose PAAI-2 where only *one* intermediate node is selected to return a report for every data packet. We remark that the strategy of selecting a subset of intermediate nodes which must return an ack tends to be vulnerable to selective dropping attacks (see §5). Consequently, we find that PAAI-2 requires more algorithmic complexity but achieves a slower detection rate than PAAI-1.

Phase 1: send data and decide whether to probe
Consider that $\mathbb{S}$ sends out a data packet $m = \langle \text{data}||\text{timestamp}\rangle$ towards $\mathbb{D}$. On receiving m, an intermediate node (including $\mathbb{D}$) first checks whether the embedded timestamp is recent. If verification fails, then m is dropped. Otherwise, $\mathbb{F}_i$ stores the identifier $\mathrm{H}(m)$ for m and starts a wait timer $t_i = r_i$. Finally, m is forwarded toward the destination.

On receiving m, $\mathbb{D}$ creates a report $\mathcal{A}_i = [\mathrm{H}(m)]_{K_d}$ and returns an ack $a_d = \langle \mathrm{H}(m)||\mathcal{A}_i\rangle$ to $\mathbb{S}$. On receiving an ack from $\mathbb{D}$ within the wait-time, an intermediate node $\mathbb{F}_i$ stores a copy of it, forwards it towards $\mathbb{S}$, and starts a waiting time $t_i = r_0 - r_i$.

If $\mathbb{S}$ receives a valid ack from $\mathbb{D}$ within a waiting time, it concludes that m arrived unaltered at $\mathbb{D}$ and the protocol is terminated for the current round. Otherwise, $\mathbb{S}$ executes the next phase of the protocol. In the following, it is implicit that a node $\mathbb{F}_i$ accepts a packet (probe or ack) iff it contains a data packet identifier already stored at $\mathbb{F}_i$.

Phase 2: probe
$\mathbb{S}$ sends out a probe $c = \langle \mathrm{H}(m)||\mathcal{Z}\rangle$ towards $\mathbb{D}$. The probe contains an identifier $\mathrm{H}(m)$ for m, and a random challenge $\mathcal{Z}$.

On receiving a probe within the wait-time[9], an intermediate node $\mathbb{F}_i$ computes a $\mathrm{PRF}_{K_i}(\cdot)$-based predicate T_i over input $\mathcal{Z}$, where T_i returns "true" with probability $\frac{1}{d-i+1}$. In what follows, we say that a node $\mathbb{F}_i$ is *sampled* for a data packet m if T_i returns true on input $\mathcal{R}$.

Finally, $\mathbb{F}_i$ starts a wait-timer $t_i = r_i$ and forwards the probe towards $\mathbb{D}$.

Phase 3: acknowledge
In this phase, the intermediate nodes must return an ack to $\mathbb{S}$. Ideally, the ack must originate at the upstream node of the link where m was dropped. To this end, we employ the following rules: (a) If an intermediate node $\mathbb{F}_i$ does not receive any ack from its downstream neighbor within the wait-time t_i, it generates an encrypted report $\mathcal{A}_i = \mathrm{E}_{K_i}([i||c||a_d]_{K_i})$.[10] (b) Otherwise, on receiving a downstream ack within the wait-time, $\mathbb{F}_i$ performs one of the following actions. If $\mathbb{F}_i$ was *sampled* for m during phase 1, it generates an encrypted report $\mathcal{A}_i$ (as described in previous case) to overwrite the report in the received ack. Otherwise it re-encrypts the report in the received ack, i.e., $\mathcal{A}_i = \mathrm{E}_{K_i}(\mathcal{A}_{i+1})$. Finally, $\mathbb{F}_i$ sends out an ack $a_i = \langle \mathrm{H}(m)||\mathcal{A}_i\rangle$ towards $\mathbb{S}$.

DEFINITION 1. *We say that a node $\mathbb{F}_e$ is **selected** for a data packet m, if (a) $\mathbb{F}_e$ is sampled for m, and (b) $\mathbb{F}_1, \ldots, \mathbb{F}_{e-1}$ are not sampled .*

From the above definition, it follows that, for a given data packet, only one intermediate node is *selected* uniformly at random with probability $\frac{1}{d}$. Observe that due to the ack forwarding mechanism described above, $\mathbb{S}$ expects an ack that was generated at the *selected* node $\mathbb{F}_e$ and re-encrypted by *each* upstream node between $\mathbb{F}_e$ and $\mathbb{S}$.

Phase 4: score
In this phase, $\mathbb{S}$ assigns numerical *scores* to the links. On receiving an ack from $\mathbb{F}_1$, $\mathbb{S}$ first decodes the embedded report $\mathcal{A}_1^m$ by performing successive decryptions using the keys $K_1, \ldots, K_e$ *in that order*, where K_e is the secret key shared between $\mathbb{S}$ and $\mathbb{F}_e$. If the final decoded value matches the expected value $\langle [e||c]_{K_e}\rangle$, then $\mathbb{S}$ decides that there was no malicious activity in the interval $[l_0, l_{e-1}]$; consequently, no scores are updated. Otherwise, $\mathbb{S}$ 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, $\mathbb{S}$ adds 1 to the individual score of each link in the interval. No scores are updated for the links in the interval $[l_e, l_{d-1}]$.

Phase 5: identify

[8] In the case where $\mathbb{S}$ does not receive any report within a wait-time, $\mathbb{S}$ can simply conclude that a drop occurred at its downstream link l_0.

[9] If the wait-timer expires, then the state maintained for m is deleted.

[10] If no ack was received from $\mathbb{D}$ in phase 1, then a_d is set to $\perp$.

$\mathbb{S}$ pre-determines a per-link drop rate threshold α, based on which it further sets a threshold ψ_{th} for the end-to-end drop rate of data packets. $\mathbb{S}$ constantly monitors the actual end-to-end data packet drop rate ψ based on the number of sent data packets and successfully received acks from the destination. It is guaranteed that $\psi_{th} < \psi$ ***iff*** there is at least one link with a drop rate exceeding α. Then the source can compute per-link loss rate based on the accumulated data and identify the link with the excessive drop rate. More details are given in §7.

7. THEORETICAL ANALYSIS

In this section, we theoretically analyze the guaranteed end-to-end throughput, detection rate, communication and storage overhead of the proposed protocols. Due to space limitations, we only provide a high-level sketch for the proofs of Theorems 1 and 2 in the appendix. The detailed proofs for all the theorems and corollaries in this paper are given in the full version available online [17]. The results are summarized in Table 1, which also gives a clear comparison between the full-ack, PAAI, and statistical FL protocol[11] [7]. In §8 we validate our theoretical results and present average-case results from simulations.

Definitions and Notation. Let ρ_i be the natural drop rate of link l_i, and suppose that ρ_i's are i.i.d. random variables with maximum value 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 ζ be the malicious end-to-end drop rate, i.e., the drop rate due to malicious links. When the observed drop rate value approaches its true value within a small uncertainty interval, the AAI false positive/negative is limited below a certain threshold ϵ. We call this the ***converged*** condition.

Let p be the probe frequency employed in PAAI-1. Further, in PAAI-2, let ψ_{th} be the threshold of the end-to-end data packet drop rate observed by the source. Let η_i be the number of times that node $\mathbb{F}_i$ is *selected* so far.

7.1 Bounding Malicious End-to-End Drop Rate

For ease of understanding, all the theoretical bounds in this subsection are computed under the converged condition. In §7.2 we derive the detection rate (number of data packets sent by the source required to reach converged condition) for the full-ack and PAAI schemes. We can see the detection rates are high in the full ack and PAAI-1 schemes, so the "unconverged" time period is negligible.

For simplicity, we first assume that an adversary employs an identical drop rate for all types of packets (data, probe or ack packets) at a controlled link l_i, and thus the probability that a packet of any kind is dropped at l_i is θ_i. Loosely speaking, the following theorem provides a bound on the damage that an adversary (with an arbitrary number of links under its control) can inflict to the network's end-to-end throughput.

THEOREM 1. *Given a path of length $\mathbb{D}$, an adversary in control of z intermediate links can cause (at most) the following malicious end-to-end drop rates without being detected: (a) $\zeta = z\alpha$ in full-ack scheme and PAAI-1, and (b) $\zeta = 1 - \frac{(1-\alpha)^{2d}}{(1-\rho)^{2(d-z)}}$ in PAAI-2 by setting the end-to-end drop rate threshold ψ_{th} as $\psi_{th} = 1 - (1-\alpha)^{2d}$.*

It is possible that an adversary may choose to drop different types of packets at different rates. However we can intuitively see that the adversary cannot gain any advantage by doing this, because dropping any type of packet will always result in an increase in the drop count of the link where the packet was dropped. The following corollary proves the statement.

COROLLARY 1. *An adversary who employs different drop rates for different types of packets achieves the same maximum end-to-end drop rate.*

Corollary 2 presents the *optimal* strategy that an adversary can employ in order to cause maximum degradation to the network throughput. The corresponding bounds on the degradation in network throughput under the optimal strategy are also presented.

COROLLARY 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 end-to-end drop rate across all the paths containing malicious links in the 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.*

7.2 Detection Rate

We compute the detection rates of the full-ack scheme and PAAIs in the following theorem.

THEOREM 2. *Given the threshold $\alpha = \rho + \epsilon$ and the allowed false positive σ, the full-ack and the PAAI protocols require the following number of packets transmitted by the source to converge. (a) $\tau_1 = \frac{ln(\frac{2}{\sigma})}{8\epsilon^2 \cdot (1-\rho)^{2+d}}$ for full-ack scheme, (b) $\tau_2 = \frac{\tau_1}{p}$ for PAAI-1, where p is the probe frequency, and (c) $\tau_3 = 2^d \frac{ln(\frac{2}{\sigma})}{18\epsilon^2} \cdot d \cdot log(d)$ for PAAI-2.*

Corollary 3 shows the sensitivity of the detection rate (achieved by the full-ack and PAAI protocols) to the various protocol parameters. As it turns out, PAAI-1 can achieve fast detection rates under various parameter settings (and thus, a wide range of empirical scenarios).

COROLLARY 3. *For both the full-ack scheme and PAAI-1, the allowed false positive rate σ is the dominating factor on their detection rates, while the network-related parameters (natural loss rate ρ and path length $\mathbb{D}$) have negligible*

[11] In the following, we compare our PAAI protocols mainly with the statistical FL protocol [7] because it is the state-of-the-art and the only protocol with a rigorous theoretical analysis, to the best of our knowledge.

influence on the detection rates. However, the detection rate of PAAI-2 heavily depends on the path length $\mathbb{D}$.

For example, if we set $\sigma = 0.03$ and $p = \frac{1}{d^2}$, and choose an arbitrary network setting where $\alpha = 0.03$, $\rho = 0.01$ and $d = 6$, then we have $\tau_1 \doteq 1500$, $\tau_2 \doteq 5 \times 10^4$ and $\tau_3 \doteq 6 \times 10^5$; whereas the detection rate in statistical FL protocol [7] is 2×10^7. Per Corollary 3, the detection rate for PAAI-1 does not vary much given other network-related parameter settings. Table 1 compares the detection rates achieved by the different protocols.

7.3 Communication Overhead

In this section we compute and compare the communication overhead incurred by the full-ack and the PAAI protocols for a given path of length $\mathbb{D}$. The analysis results are presented in Table 1.

Full-ack. Recall from §4 that in the benign case where *no* packet loss occurs, each data packet requires one $O(1)$-sized ack from the destination. When a packet drop happens, the source solicits a $O(d)$-sized onion report via a $O(1)$-sized probe packet. Therefore, given the end-to-end loss rate ψ, the overall communication overhead per packet is $O(1+d\psi)$. $H(m)$ using the secret key K_d shared with $\mathbb{S}$, and sends out an ack $a_d = \langle H(m)||\mathcal{A}_d\rangle$. On receiving an ack a_{i+1} from its downstream neighbor, an intermediate node $\mathbb{F}_i$ "wraps" the ack in an ***onion*** fashion: it extracts the report $\mathcal{A}_{i+1}$ and then computes its own report as a MAC over the concatenation of the identifier $H(m)$ and the downstream report $\mathcal{A}_{i+1}$ using the secret key K_i shared with $\mathbb{S}$. It then sends out an ack $a_i = \langle H(m)||\mathcal{A}_{i+1}||[H(m)||\mathcal{A}_{i+1}]_{K_i}\rangle$ towards $\mathbb{S}$. However, if no ack is received within the wait-time, $\mathbb{F}_i$ creates a fresh ack in a manner similar to the base station and sends it towards $\mathbb{S}$. Now, on receiving the final ack, $\mathbb{S}$ can sequentially verify each report embedded in it. For some $i < d$, if the MAC from each intermediate node $\mathbb{F}_j, j \in [1, i]$ is valid but the MAC from $\mathbb{F}_{i+1}$ is invalid or not present in the final ack, then $\mathbb{S}$ concludes that either the data packet m, or an **PAAI-1.** Recall from §6.1 that for each sampled data packet, the source solicits one $O(d)$-sized onion report (in case of authenticated probes, the size of a probe packet is also $O(d)$). Since a given data packet is sampled only with probability p, the amortized communication overhead per data packet is $O(pd)$. By setting $p = \frac{1}{d^2}$ we can get $O(\frac{1}{d})$ overall communication overhead per packet. Note that the above results apply *regardless of whether there are packet drops or not*.

PAAI-2. Recall from §6.2 that each intermediate node $\mathbb{F}_i$ on the forwarding path either generates a new ack or re-encrypts the ack received from downstream. Therefore, an ack packet traversing the path has a constant size ($O(1)$) at any point in time. Further, PAAI-2 requires one $O(1)$-sized probe packet per data packet sent by the source. Note that the above results apply *regardless of whether there are packet drops or not*.

7.4 Storage Overhead

Storage is a major concern in certain resource-constrained networks. An adversary may even exploit the storage limitation and manipulate packet drops to intentionally create the worst case condition for the storage overhead of an AAI protocol. On the other hand, in practical settings, including when the adversary has been identified (and bypassed), excessive packet dropping is infrequent (thus the worst cases do not arise frequently). A high storage overhead in such an *ideal* case is undesirable. Therefore, in this section we analyze and compare the storage overhead in both worst and ideal cases for the full-ack scheme and PAAIs. In §8 we present the average-case storage overhead via simulations.

In the following, let ν be the number of data packets that $\mathbb{S}$ sends out per unit time. Recall that r_i denotes the round trip time between node $\mathbb{F}_i$ and $\mathbb{D}$. The results given below are summarized in Table 1.

Full-ack. In the worst case, on receiving a data packet m, an intermediate node $\mathbb{F}_i$ needs to first wait r_0 time for a probe from the source, and r_i time for an ack from $\mathbb{F}_{i+1}$. Therefore $\mathbb{F}_i$ can at most store $O(2r_0\nu)$ packets at a time. In the ideal case without packet drop, $\mathbb{F}_i$ only needs to store a packet for r_i time before receiving an ack from $\mathbb{F}_{i+1}$.

PAAI-1. If a data packet m is not selected for a probe, $\mathbb{F}_i$ needs to wait $\frac{r_0}{2}$ time for a probe packet from the source. If m is selected for a probe, in the worst case $\mathbb{F}_i$ needs to further wait r_i time for an ack from $\mathbb{F}_{i+1}$; whereas in the ideal case, $\mathbb{F}_i$ needs to further wait r_i time for the ack from $\mathbb{F}_{i+1}$. Therefore given the probe frequency p, $\mathbb{F}_i$ can at most store $(0.5 + p)r_0 \times \nu$ packets at a time in both the worst and ideal cases.

PAAI-2. In the worst case, on receiving m, $\mathbb{F}_i$ waits r_i time for an ack from $\mathbb{F}_{i+1}$, $r_0 - r_i$ time for a probe from the source, and r_i time for an ack from $\mathbb{F}_{i+1}$ again, which gives the worst case storage overhead $0(2r_0\nu)$. In the ideal case, $\mathbb{F}_i$ only needs to wait r_i time for the ack from $\mathbb{F}_{i+1}$. Therefore in ideal case the storage bound is $O(r_0 \times \nu)$ packets at a time.

8. SIMULATION RESULTS AND ANALYSIS

We implement a simulator to study the *average-case* performance of the proposed protocols, and also contrast the average-case results with the theoretical results (as listed in Table 2). Through simulations, we not only validate our theoretical results and make comparisons, but also derive new observations missing from theoretical analysis by itself.

8.1 Methodology

Adversary. Note that, in practice, an adversary usually directly compromises a *node*, dropping the traffic flowing through that node at the adversary's will. We emulate such a realistic scenario by setting malicious *nodes* in the path to perform malicious packet dropping activity. We simulate the adversary's *optimal strategy* by deploying exactly one malicious node on the path (Corollary 1). Recall that, in our protocols, if a malicious node drops packets, it can manifest high drop rates only on its adjacent links. We also set the adver-

Protocol	Detection Rate	Communication	Storage	
			worst	ideal
Full-ack	$\frac{ln(\frac{2}{\sigma})}{8\epsilon^2\cdot(1-\rho)^{2+d}}$	$O(1+\psi d)$	$O(2r_0\nu)$	$O(r_0\nu)$
PAAI-1	$p\frac{ln(\frac{2}{\sigma})}{8\epsilon^2\cdot(1-\rho)^{2+d}}$	$O(pd)$	$O(r_0(0.5+p)\nu)$	$O(r_0(0.5+p)\nu)$
PAAI-2	$2^d\frac{ln(\frac{2}{\sigma})}{18\epsilon^2}\cdot d\cdot log(d)$	$O(1)$	$O(2r_0\nu)$	$O(r_0\nu)$
Statistical FL [7]	$d^2\frac{ln\frac{d}{\sigma}}{p\epsilon^2}$	$O(\frac{p\epsilon^2}{dln\frac{d}{\sigma}})$	$O(pr_0\nu)$	$O(pr_0\nu)$
Combination 1	$p\frac{ln(\frac{2}{\sigma})}{8\epsilon^2\cdot(1-\rho)^{2+d}}$	$O(p(1+\psi d))$	$O(r_0(0.5+2p)\nu)$	$O(r_0(0.5+2p)\nu)$
Combination 2	$2^d\frac{ln(\frac{2}{\sigma})}{18\epsilon^2\times p}\cdot d\cdot log(d)$	$O(p)$	$O(r_0(1+p)\nu)$	$O(r_0\nu)$

Table 1: Detection rate and overhead comparison. The notation is given at the beginning of §7. We translate the related results [7] using our notation. Combination 1 and Combination 2 are described in §10.

sary to employ the following tactics: (a) Since the full-ack scheme and PAAI protocols ensure that the adversary cannot gain benefit by dropping different packets at different rates (Corollary 1), the adversary drops all types of packets at the same rate. (b) Without loss of generality, we assume that, when the malicious node receives but drops a data packet, on receiving an ack request it will still send back the ack as if it were functioning correctly. In this way, a malicious node $\mathbb{F}_i$'s dropping activity always increases the drop counts of its downstream adjacent link l_i. Therefore l_i is the target to identify.

Topology and Parameters. Recall the example topology given in Figure 1. We simulate the proposed protocols on one path with various lengths and varying locations of the malicious link. Due to lack of space, here we only present the results for an arbitrary setting where $d = 6$ and $\mathbb{F}_4$ is set to be the node controlled by the adversary (results from other settings present similar trends and conclusions). According to our aforementioned adversarial setting, the malicious drops will directly increase l_4's drop count; thus l_4 is the target link for our AAI protocols to identify. In the following we also call l_4 as l_M. We follow the example parameter settings used in our previous theoretical analysis, i.e., we set benign per-link loss rate threshold $\rho = 0.01$ and malicious per-link drop rate $\alpha = 0.03$ (we implement this by setting a drop rate of 0.02 for the malicious *node* $\mathbb{F}_4$). However, recall from Corollary 3 and Table 1 that the performance of PAAI-1 does not degrade in the case of longer paths and higher natural loss rates. Each packet traversing a link (or the malicious node) has an independent probability of being dropped bi-directionally below the corresponding drop rate threshold of that link (or the malicious node). We also set per-link bi-directional latency distributed within 0 to 5 ms uniformly at random.

Evaluation Metrics. We evaluate (a) AAI false positive and negative rates (which directly reflect detection rates) and (b) storage overhead of each node for the full-ack and PAAI protocols.[12] we run the simulation 10000 times for each protocol to calculate the false positive and false negative rates and plot their dynamics over time. Recall from Table 1 that storage overhead directly depends on packet origination rate; as such we evaluate it for different orders of origination rate: 1000 and 100 data packets per second (the storage overhead under a source's sending rate of 10 packets per second is too low to exhibit any insightful traits).

[12] We did not simulate the communication overhead because the theoretical analysis already gives straightforward and tightly bounded results.

Protocol	Detection Rate (minutes)		Storage (# pkt)	
	bound	average	bound	average
Full-ack	0.25	0.17	12	3.2
PAAI-1	9	4.2	3.2	3.0
PAAI-2	100	50	12	6.4
Statistical FL [7]	3333	N/A	< 1	N/A

Table 2: Comparison of detection rates between theoretical results and simulation results. The source's sending rate is set to 100 data packets per second. The storage overhead is the average number of packets stored in $\mathbb{F}_1$ with the presence of a malicious link l_4.

8.2 Results and Analysis

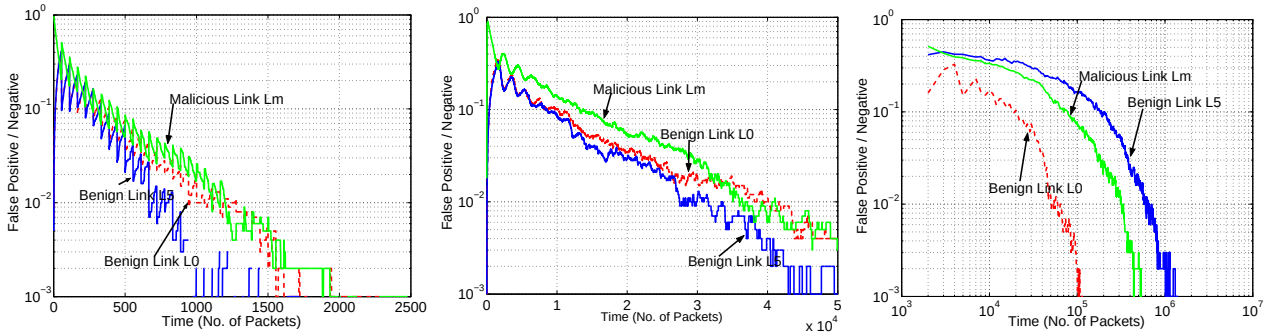
As presented below, we are able to both validate our theoretical results and to derive new and interesting observations from the simulation results.

8.2.1 False Positive and Negative

Figure 2 plots the false positive and false negative rates observed from 10000 simulation runs for each protocol. From the figure we can observe that, given the same false positive threshold $\sigma = 0.03$, the detection rates are nearly twice of the corresponding theoretical bounds. We summarize the comparisons between theoretical and experimental results in Table 2. In addition, we can see that in PAAI-2, the source takes more time to accurately observe the per-link drop rate for a link farther away from the source. This fact can be theoretically proved via the mathematical formula (we defer the proof to the full version).

8.2.2 Storage Overhead

We launch two different sets of simulations to study the characteristics of storage overhead in AAI protocols. In each

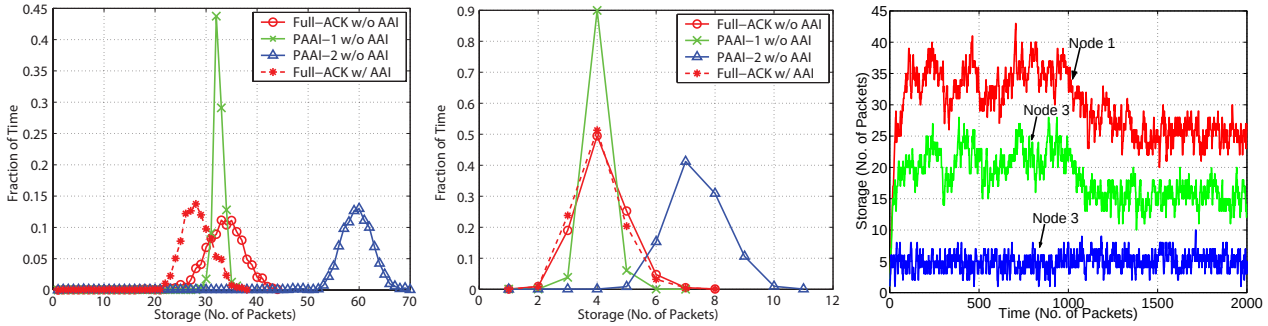


(a) Full-ack scheme. We use logarithmic scale for the y-axis.

(b) PAAI-1. We use logarithmic scale for the y-axis.

(c) PAAI-2. We use logarithmic scale for both axes.

Figure 2: False positive and negative. The time is measured by the number of packets sent by the source.



(a) Sending rate = 1000 pkt/sec.

(b) Sending rate = 100 pkt/sec.

(c) Storage traits with sending rate = 1000 pkt/sec in full-ack scheme.

Figure 3: Storage overhead. The storage is measured by the number of packets stored at any given time.

scenario if an AAI protocol reaches the converged condition (after 10^3, 2.5×10^4 and 3×10^5 data packets sent by the source in full-ack, PAAI-1 and PAAI-2 schemes, respectively), we assume the source bypasses the identified l_4 by replacing $\mathbb{F}_4$ with a honest node $\mathbb{F}_4'$ to connect nodes $\mathbb{F}_3$ and $\mathbb{F}_5$ (we implement this in the simulation by resetting $\mathbb{F}_4$'s drop rate to zero). We label cases where adversary identification comes into play as "w/ AAI". We also simulate the case where the existing adversary is not identified and bypassed, which is labeled as "w/o AAI".

We first investigate the storage overhead of a single node $\mathbb{F}_1$ (which has the highest storage overhead, as we show later) under different source's sending rates (1000 and 100 data packets per second). We first let the source send 2000 data packets in total, within which only the full-ack scheme can reach the converged condition. However, we present the results for the full-ack scheme in both "w/ AAI" and "w/o AAI" cases to compare with the PAAI protocols. Figures 3(a) and 3(b) present $\mathbb{F}_1$'s storage overhead when the source's sending rate is 1000 or 100 data packets per second, respectively. It is apparent that the storage overhead decreases with the lower sending rate. We further observe that, in the "w/o AAI" case, PAAI-1 possesses the lowest storage overhead; and the storage overhead of each protocol increases roughly linearly with the source's sending rate. This fact complies with our theoretical bounds (Table 1). In addition, it is clear that the full-ack scheme achieves a lower storage overhead after bypassing the adversary ("w/ AAI"). Therefore, though the full-ack scheme presents the highest theoretical bound of worst-case storage overhead, it achieves the lowest storage overhead in practice when AAI comes into play. This observation implies that, in essence, a protocol with a higher detection rate benefits more from ideal cases where packet drops are rare after the adversary is quickly bypassed.

In another simulation, we investigate the storage overhead of nodes at different locations in the path and the influence of AAI on storage overhead. Since the full-ack scheme has the highest detection rate, we only present the simulation results of the full-ack scheme due to space limitations (the results derived from other protocols present common trends). To make the influence of AAI more graphically obvious, we enlarge the drop rate of $\mathbb{F}_4$ to 0.1. In this simulation we let the source send 2000 data packets at the rate of 1000 data packets per second, and bypass the adversary after sending 1000 data packets. Figure 3(c) plots the resulting dynamics of the storage overhead of nodes $\mathbb{F}_1$, $\mathbb{F}_3$ and $\mathbb{F}_5$, from which we can observe that, nodes closer to the destination have lower storage overhead and are less affected after adversarial packet drops. This observation can be explained according

to the theoretical analysis in §7.4.

9. SUMMARY OF RESULTS

From the theoretical and experimental results presented before, we can make the following major observations:

Theory vs. Simulation. The average-case results derived from our simulations are within the corresponding theoretical bounds. For the detection rate, the average-case results are nearly two times better than the corresponding theoretical results. For the storage overhead, the average-case result of the full-ack scheme is far smaller than its worst-case bound, thanks to its fast convergence. The PAAI-1 protocol also presents low storage overhead, even with the presence of an adversary.

Practicality. We make the following conclusions about the trade-off between the three performance metrics achieved by the various protocols: (a) The full-ack scheme offers the fastest detection rate and incurs a low storage overhead, but at the cost of impractical communication overhead. (b) The PAAI-1 protocol offers a practical (though not the best) detection rate and communication and storage overhead simultaneously. More specifically, given that each data packet is 1.5KB (which is the currently popular MTU standard), per Figures 3(a) and 3(b), PAAI-1 introduces less than 45KB additional storage overhead even at its peak value under an intense packet sending rate of 1.5MB per second, and around 6KB at its peak value under a packet sending rate of 150KB per second. Furthermore, by setting the sampling rate $p = \frac{1}{5d^2}$, PAAI-1 poses only around 3% additional communication overhead in a path with length $d = 6$, per Table 1; while the detection rate is 45 minutes given by the theoretical bound, and around 20 minutes on average per Table 2 (in previous analysis and simulation we set $p = \frac{1}{d^2}$). (c) The PAAI-2 protocol presents worse performance compared to the full-ack scheme and PAAI-1 protocol, but still presents a more practical detection rate compared to the statistical FL scheme [7] (see below). (d) The statistical FL protocol [7] incurs almost optimal communication and storage overhead, but achieves a rather impractical detection rate – nearly 50 hours in the worst case (Table 2). We conclude that PAAI-1 offers the most desirable trade-off between the performance metrics. In contrast, all the other protocols only optimize at most two performance metrics at the cost of deteriorating the other metric(s) undesirably.

10. COMBINATION

So far we have explored three different basic approaches, namely: (a) every node acknowledges every lost data packet (exemplified by the full-ack scheme), (b) every node acknowledges a selected fraction of data packets (instantiated by the PAAI-1 protocol), and (c) a selected subset of nodes acknowledge every data packet (represented by the PAAI-2 protocol). Intuitively, it might be tempting to consider combinations of the above basic approaches in order to improve upon a certain performance metric. However, as we demonstrate below, the combinations may not necessarily achieve a better trade-off between the performance metrics as compared to the basic approaches, and may therefore be unfavorable in practice. Specifically, although a combination may further optimize a certain performance metric, other metrics can degrade undesirably at the same time. Due to lack of space, we will briefly discuss two sample combinations and analyze the corresponding tradeoff.

Combination 1. By combining the basic approaches (a) and (b) above, we can design a protocol where every node must acknowledge a selected fraction of *lost* data packets. The PAAI-1 protocol can be easily modified to follow the above approach. Specifically, instead of using a secret key known only to $\mathbb{S}$ to implement the probe function, we will use the secret key K_d shared between $\mathbb{S}$ and $\mathbb{D}$. Now, on receiving a data packet, $\mathbb{D}$ can independently decide whether it must be acknowledged. For a sampled data packet m, $\mathbb{S}$ will send out a probe only if it fails to receive an ack from $\mathbb{D}$. The remaining details follow from PAAI-1. While retaining the same detection rate as PAAI-1, the new protocol further reduces the communication overhead, since $\mathbb{S}$ now solicits an onion report for only a *lost* sampled packet (instead of every sampled packet in PAAI-1). However, the storage overhead increases: in the worst case, on receiving m, each node must first wait an additional r_0 time for an ack from $\mathbb{D}$, such waiting time which was not required in PAAI-1. Its performance is summarized in Table 1.

Combination 2. By combining the basic approaches (b) and (c) above, we can design a protocol where one *selected* node acknowledges a selected fraction of data packets. Similar to Combination 1, we will use a probe function that is implemented using the secret key K_d. The data packet structure will be similar to that in PAAI-2. Now, on receiving a data packet, $\mathbb{D}$ can independently decide whether it must be acknowledged. If an intermediate node receives a *valid* ack from $\mathbb{D}$, it immediately knows that the packet was sampled and that there will be no further probe. For a sampled data packet, $\mathbb{S}$ will send out a probe only if it fails to receive an ack from $\mathbb{D}$. The remaining details follow from PAAI-2. It is intuitive to see the new protocol incurs lower communication overhead than both PAAI-1 and PAAI-2, but at the price of a lower detection rate. Its performance is summarized in Table 1.

11. CONCLUSION

In this paper, we address the problem of designing a secure AAI protocol that offers a *practical* trade-off between the three essential performance metrics: detection rate, communication overhead, and storage overhead. To this end, we systematically explore the design space of AAI protocols, and propose a set of basic protocols where each protocol exemplifies a design dimension. Based on our theoretical analysis and simulation results, we conclude that the proposed PAAI-1 protocol achieves the best trade-off, and as a result is more practical than the other protocols.

We note, however, that as a first step toward a systematic exploration of the design space for AAI protocols, this

paper inevitably bears some limitations in its extensibility and generality. The quest for an *optimal* AAI protocol still remains an open problem. Finally, we note that our PAAI protocols require the additional assumption of loose time-synchronization, which, although a viable assumption for many network settings, might limit their applicability.

12. ACKNOWLEDGEMENTS

The authors gratefully thank Haowen Chan for constructive discussions and insightful suggestions, and the anonymous reviewers for their valuable feedback.

13. REFERENCES

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

[2] 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.

[3] 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`.

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

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

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

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

[8] 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.

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

[10] 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.

[11] M. Just, E. Kranakis, and W. Tao. Resisting malicious packet dropping in wireless ad hoc networks. In *Proceedings of ADHOC-NOW*, Oct. 2003.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] X. Zhang, A. Jain, and A. Perrig. Full version: Packet-dropping adversary identification for data plane security. Available at `http://www.cs.cmu.edu/~xzhang1/doc/conext08_full.pdf`.

APPENDIX

Due to space limitations, we only provide a high-level summary of proofs for Theorems 1 and 2. Detailed proofs for all the theorems and corollaries in the paper body are contained in the full version available online [17].

Proof of Theorem 1: In the following, we analyze the full-ack and PAAI-1 protocols together, and PAAI-2 separately.

I. Full-ack and PAAI-1. Since the onion report used in the full-ack and PAAI-1 schemes can be used to locate a specific link for each lost packet, *under converged condition* each malicious link can at most drop α fraction of packets without being detected. This in turn implies our results in Theorem 1 for the full-ack and PAAI-1 schemes.

II. PAAI-2. First we establish an end-to-end drop rate threshold ψ_{th} so 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). If each link l_i has a drop rate $\theta_i < \alpha$, we can further compute the end-to-end drop rate ψ_d. Then we compute the maximum end-to-end drop rate that an adversary can cause without being detected, i.e., without causing $\psi_d > \psi_{th}$. ■

Proof of Theorem 2: In the 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. We mainly leverage the *Maximum Likelihood Estimation* and *Hoeffding's inequality* for the above calculation. ■