Dealing with Liars: Misbehavior Identification via Rényi-Ulam Games*

William Kozma Jr. and Loukas Lazos

The University of Arizona, Electrical and Computer Engineering Dept. Tucson, Arizona, 85712
{wkozma,llazos}@ece.arizona.edu

Abstract. We address the problem of identifying misbehaving nodes that refuse to forward packets in wireless multi-hop networks. We map the process of locating the misbehaving nodes to the classic Rényi-Ulam game of 20 questions. Compared to previous methods, our mapping allows the evaluation of node behavior on a per-packet basis, without the need for energy-expensive overhearing techniques or intensive acknowledgment schemes. Furthermore, it copes with colluding adversaries that coordinate their behavioral patterns to avoid identification and frame honest nodes. We show via simulations that our algorithms reduce the communication overhead for identifying misbehaving nodes by at least one order of magnitude compared to other methods, while increasing the identification delay logarithmically with the path size.

1 Introduction

Multi-hop networks, such as wireless ad-hoc, sensor, and mesh networks rely on collaboration among network nodes to provide reliable data services. If the destination is not within the communication range of the source, data has to be relayed by intermediate nodes. Implicit in this relay process is the assumption that intermediate nodes are willing to forward traffic other than their own.

However, a fraction of nodes may not conform to the specifications of collaborative routing protocols. Sophisticated users can misconfigure their devices to behave in a selfish manner and drop relay traffic, in order to save energy resources [8, 9, 34]. Moreover, in hostile environments, an adversary may compromise several nodes and configure them to misbehave. It has been shown that even a small fraction of misbehaving nodes refusing to relay packets, can lead to a significant drop in the overall network performance [6, 7, 20, 21]. In this paper, we address the problem of *developing resource-efficient methods for identifying nodes that refuse to collaborate in relaying packets.* We define resource efficiency in terms of the communication overhead associated with the identification of all misbehaving nodes along a routing path.

Previously proposed solutions addressing routing misbehavior can be classified to reputation-based systems [6, 7, 21], credit-based systems [8, 9, 16, 34],

* This research was supported by BAE systems, and Connection One (an I/UCRC NSF/industry/university consortium).

Y. Chen et al. (Eds.): SecureComm 2009, LNICST 19, pp. 207–227, 2009.

and acknowledgment-based systems [1, 2, 18, 20, 23]. A common element in all these solutions is the evaluation of node behavior on a *per-packet* basis. This approach provides a fine granularity in quantifying the behavior of nodes and low delay in identifying the misbehaving ones. However, it expends energy (in the form of receptions or transmissions) on a per-packet basis. For example, in acknowledgment-based systems, packets must be acknowledged two or more hops upstream [2, 1], thus consuming energy and bandwidth.

We develop a communication-efficient solution that allows the per-packet evaluation of behavior while not incurring the per-packet overhead. Nodes themselves are responsible for monitoring the packets they receive and forward to the next hop. When misbehavior is observed on a particular path, the source requests from nodes along the path to commit to a proof of the packets they receive and forward via an audit process (similar to [18, 1]). Although misbehaving nodes may lie about the packets they forward, the source combines multiple audit replies from honest nodes to identify the misbehaving ones.

Our Contributions: We map the problem of misbehavior identification to the classic Rényi-Ulam game of 20 questions [29, 32]. Rényi-Ulam games have been extensively used in various contexts including error correction codes [3], selecting, sorting, and searching in the presence of errors [25, 30, 31], to name a few. We develop communication-efficient algorithms for locating misbehaving nodes, based on different versions of Rényi-Ulam games. Our mapping allows the per-packet evaluation of node behavior without incurring the per-packet communication overhead. Furthermore, our formulation addresses colluding adversaries who coordinate their attacks to avoid identification and frame honest nodes.

The remainder of the paper is organized as follows. In Section 2, we present related work. In Section 3, we state the problem and our model assumptions. In Section 4, we map the misbehavior identification problem to Rényi-Ulam games and develop two auditing (searching) strategies. In Section 5, we present an efficient method for constructing audits. In Section 6, we compare the performance of our algorithms to previously proposed schemes. In Section 7, we conclude.

2 Related Work

Previously proposed methods for addressing the misbehavior problem can be classified into three categories: (a) credit-based systems, e.g., [8, 9, 16, 34], (b) reputation-based systems, e.g., [14,7,13,21,6,22], and (c) acknowledgment-based systems, e.g., [1, 2, 20, 23].

Credit-Based Systems: Credit-based systems [8,34,9,16] are designed to provide incentives for forwarding packets in the form of credit payments. Nodes accumulate credit that can be later used to pay for sending their own traffic. Buttyan et al. [8, 9] proposed a scheme in which a *nuglet counter* is used to tabulate the amount of credit accumulated at each node. To prevent tampering with the accumulated credit, the nuglet counter is implemented in tamper proof hardware. Zhong et al. [34] proposed Sprite, in which nodes collect *receipts* for the packets

they forward which can be later exchanged for credit in a Credit Clearance Service (CCS). Jakobsson et al. [16] used cryptographic payment tokens that are attached to all packets and managed by a virtual bank. In credit-based systems a misbehaving node can drop relayed traffic if it is not interested in routing its own packets. Moreover, colluding nodes can agree to forward their own flows to accumulate credit while dropping all other flows. Finally, credit-based systems favor well connected nodes to boundary ones.

Reputation-Based Systems: Reputation-based systems [6,7,13,21,22,14] rely on building a reputation metric for each node according to its behavioral pattern. Buchegger et al. [6,7] proposed the *CONFIDANT* scheme, in which neighboring nodes monitor the behavior of their peers via overhearing. A similar monitoring method was proposed by Marti et al. [21]. In building the reputation metric, monitoring nodes usually overhear the transmission and reception of messages on a per-packet basis, thus operating their radio in promiscuous mode. Ganeriwal et al. [13] used a Bayesian model to map binary ratings into reputation metrics. He et al. [14] proposed SORI, which monitors neighboring nodes using a watchdog mechanism and propagates collected information to nearby nodes, thus relying on both first- and second-hand evaluations. Michiardi et al. [22] proposed CORE, where nodes combine reports from other nodes and task-specific monitoring to assign reputation metrics.

Node monitoring becomes complex in cases of multi-channel networks or nodes equipped with directional antennas. Neighboring nodes may be engaged in parallel transmissions in orthogonal channels thus being unable to monitor their peers. Moreover, operating in promiscuous mode requires up to 0.5 times the amount of energy for transmitting a message [12], thus making message overhearing an energy expensive operation.

Acknowledgment-Based Systems: Acknowledgment-based systems [1, 2, 20, 23] rely on the reception of acknowledgments to verify that a message was forwarded to the next hop. Liu et al. [20] proposed the 2ACK scheme, where nodes explicitly send acknowledgments two hops upstream to verify cooperation. Packets that have not yet been verified remain in a cache until they expire. A value is assigned to the quantity/frequency of unverified packets to determine misbehavior. The 2ACK scheme is susceptible to collusion of two or more consecutive nodes. Furthermore, colluding nodes can frame honest ones by claiming not to receive the acknowledgments. Padmanabhan et al. [23] proposed a method based on traceroute in which the source probes the path with pilot packets indistinguishable from data packets. Finally, Awerbuch et. al. [1] proposed an ACK-based scheme relying on a binary search process to identify a single misbehaving link. As with previous schemes, node collusion is not considered.

In our previous work [18], we proposed REAct, a reactive misbehavior identification scheme relying on audits. In REAct, the destination periodically sends acknowledgments to the source indicating the performance on the route. In the case of a performance drop, the source initiates a series of random audits to identify the misbehaving nodes. Nodes in the path in question provide a proof of

the packets they forward to the next hop using Bloom filters. REAct reduces the communication overhead for identifying misbehaving nodes due to the compact representation of its audits. However, REAct does not address collusion.

3 Network and Adversarial Models

Network Model: We assume a multi-hop ad hoc network where nodes collaboratively relay traffic according to an underlying routing protocol such as DSR [17] or AODV [26]. The path P_{SD} used to route traffic from a source S to a destination D is assumed to be known to S. This is true for source routing protocols such as DSR. If DSR is not used, P_{SD} can be identified through a traceroute operation. For simplicity, we number nodes in P_{SD} in ascending order, i.e., n_i is *upstream* of n_j if $i < j$.

We assume that the source and destination collaboratively monitor the performance of P_{SD}. The destination periodically reports to the source critical metrics such as throughput or delay. If a misbehaving node drops the periodic updates as part of its misbehavior pattern, the source interprets the lack of updates as misbehavior. Likewise, the destination explicitly alerts the source in case the performance in P_{SD} is restored. These alerts are used to pause the misbehavior identification process and account for: (a) temporal variations of performance due to traffic or intermittent connectivity, and (b) random behavioral patterns of the misbehaving nodes. We initially consider a quasi-static network in which P_{SD} does not change during the misbehavior identification process. This is later relaxed, allowing changes in P_{SD} due to node mobility.

We assume that the integrity, authenticity, and freshness of critical control messages can be verified using resource-efficient cryptographic methods. For example, a public key cryptosystem realized via computationally-efficient elliptic curve cryptography may be used to verify the authenticity and integrity of messages while providing confidentiality [19]. Note that such cryptosystems require the existence of a trusted certificate authority (CA) for initialization (issuance of keys and certificates) as well as revocation of users via a certificate revocation list (CRL). Several methods have been proposed for the distributed implementation of a CA [11, 28, 33]. Alternatively, methods based on symmetric keys can be used to protect critical messages [15, 24, 27].

Adversarial Model: We assume that a set M of misbehaving nodes exist in a path of length $k \geq |M|$. Misbehaving nodes can be located anywhere in P_{SD}. The source and destination have a mutual interest in communicating, thus misbehavior of S and D is not considered. Misbehaving nodes are aware of the mechanism used for misbehavior identification. The goal of misbehaving is twofold; degrade throughput in P_{SD}, and remain undetected. We consider two models with respect to the behavioral pattern of nodes in M.

Independently misbehaving nodes: In this model, nodes in P_{SD} misbehave independently without coordinating their packet dropping patterns. Misbehavior is modeled after an ON/OFF process in which nodes alternate between dropping

packets and behaving honestly. The duration of the misbehaving/behaving period is exponentially distributed with parameters μ_1, μ_2.

Colluding nodes: Colluding nodes share information with respect to the misbehavior identification process. For example, one misbehaving node can notify another of any actions of the source. Information sharing is achieved either in-band via the exchange of encrypted messages, or through an out-of-band coordination channel. Based on collective knowledge, the colluding nodes coordinate their behavioral patterns to avoid identification or frame honest nodes. In this model, we assume that colluding nodes are controlled by a single entity.

4 Misbehavior Identification

4.1 Motivation and Problem Mapping

The behavior monitoring mechanisms in previously proposed schemes operate on a per-packet basis, either with acknowledgments [1, 2, 20, 23], or message overhearing [6, 7, 21]. To reduce this overhead, we request nodes to self-evaluate the set of packets they forward to the next hop. In this self-evaluation process, honest nodes faithfully report the set of packets they received and forwarded, while misbehaving nodes may lie regarding packets they dropped.

We map the process of identifying lies to Rényi-Ulam searching games [29,32], that have been used for recovering an unknown value in the presence of errors. Using our mapping to Rényi-Ulam games, we develop novel misbehavior identification methods that are collusion resistant. We first provide a brief background on Rényi-Ulam games and then describe our mapping.

Background on Rényi-Ulam Games: Rényi-Ulam games are searching games independently proposed by Rényi [29] and Ulam [32]. These games involve two players; a questioner and a responder. The responder selects a secret value ω from a finite search space Ω. The questioner attempts to determine ω by asking at most q questions to which the responder is allowed up to ℓ lies. Before starting the game, the players agree on: (a) the search space Ω, (b) the number of questions q, (c) the number of lies ℓ, and (d) the mode of interaction between the players. The format of the questions can be classified into three categories: (a) bit questions, (b) cut questions, and (c) membership questions. Bit questions are defined as "Is the ith-bit of ω equal to 1?" Cut questions are defined as, for some $y \in \Omega$, "Is $\omega \leq y$?" Membership questions are defined as, for some subset $A \subseteq \Omega$, "Is $\omega \in A$?" The same questioning format is assumed for the entire game.

Two modes are possible for the interaction between the players; batch mode and adaptive mode. In batch mode, the questioner submits all questions to the responder at the same time. The responder is therefore able to review all questions before answering. In adaptive mode, the questioner asks questions one at a time. The questioner can adapt its strategy based on all previous answers. The questioner wins the game if it determines ω after at most q questions. Else, the responder wins. The questioner is said to have a "winning strategy" if it can find ω after at most q questions, independent of ω, or how the responder lies.

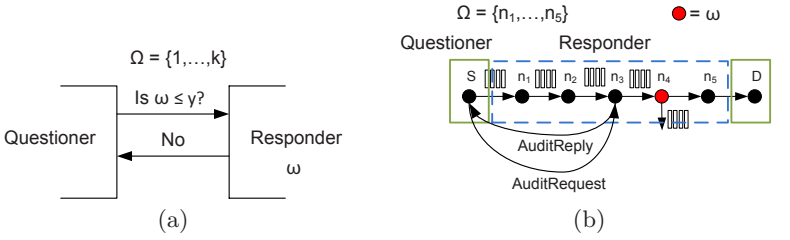

Fig. 1. (a) A generic Rényi-Ulam game. (b) Misbehavior identification mapped to a Rényi-Ulam game.

Mapping to Rényi-Ulam Games: In our mapping of misbehavior identification to Rényi-Ulam games, the role of the questioner is assumed by the source and destination, while the role of the responder is assumed by P_{SD}. The search space is defined as the set of nodes in P_{SD}, i.e., $\Omega = \{n_1, \ldots, n_k\}, k = |P_{SD}|$. The responder selects $\omega \in \{1, \ldots, k\}$, corresponding to the node n_ω in P_{SD} which is misbehaving. The source's goal is to determine n_ω, i.e., to locate the misbehaving node. Questions submitted by the questioner correspond to audits performed by the source to nodes in P_{SD}.

When responding to an audit, nodes state the set of packets forwarded to the next hop. The source combines one or more audits to construct bit, cut, or membership questions. The responder lies when a misbehaving node lies with respect to the packets forwarded to the next hop. For example, a node lies by either claiming to forward all packets received when in reality it drops them, or claiming to have forwarded no packets indicating they were dropped somewhere upstream. The location of the misbehaving nodes in P_{SD} is mapped to the placement of such lies by the responder. Note that since the responder is a single entity controlling the lies (i.e. location of misbehaving nodes and response to audits), our mapping implicitly assumes collusion. Figures 1(a) and 1(b) show the mapping of the misbehavior identification problem to a Rényi-Ulam game.

In our game, an honest node will always respond faithfully to an audit, thus a lie can only occur if a misbehaving node is audited. By adaptively selecting the nodes to be audited, the source can gather sufficient honest replies to identify nodes in M. If each node in P_{SD} is audited at most one time, the number of possible lies is limited to $\ell = |M|$. If nodes are audited multiple times, the number of lies depends on the exact auditing strategy. We now present two adaptive auditing strategies inspired by Rényi-Ulam games.

4.2 Rényi-Ulam Inspired Auditing Strategies

Let X_i denote the set of packets forwarded by a node n_i to the next hop. For example, the source sends packets X_S to the destination, and nodes n_i, n_j forward packets X_i, X_j respectively. In the absence of misbehavior in P_{SD} and assuming no packet loss $X_S = X_i = X_j$. In reality, some portion of the packets may be lost due to the wireless channel conditions or congestion, and hence $X_S \approx X_i \approx X_j$.

Definition 1. *A link* (n_i, n_{i+1}) *is defined as misbehaving if its two incident nodes* n_i, n_{i+1} *provide conflicting claims with respect to the packets forwarded to the next hop, i.e.,* $|X_i \bigcap X_{i+1}| \ll |X_i|$.

Proposition 1. *At least one node incident to a misbehaving link is misbehaving.*

Proof. By contradiction. Assume that both nodes n_i, n_{i+1} of a misbehaving link are honest. Hence, the set of packets X_{i+1} forwarded by n_{i+1} to the next hop is approximately equal to the set of packets X_i, forwarded by n_i to n_{i+1}, i.e., $|X_i \bigcap X_{i+1}| \approx |X_i|$. This contradicts the definition of a misbehaving link.

Definition 2. *A simultaneous audit is defined as auditing two or more nodes with respect to the same set of packets* X_S*, sent from* S *to* D *via* P_{SD}.

Corollary 1. *The link between two behaving nodes* n_i, n_{i+1} *cannot be identified as misbehaving, when* n_i, n_{i+1} *are simultaneously audited.*

Proof. By Proposition 1, at least one misbehaving node is incident to any misbehaving link. Hence, two behaving adjacent nodes cannot be incident to a misbehaving link. The simultaneous audit requirement ensures that the dropping pattern of any misbehaving node upstream of behaving node n_i has the same effect on the packets observed by n_i, n_{i+1}. Thus packets forwarded by n_i are also forwarded by n_{i+1}, i.e., $|X_i \bigcap X_{i+1}| \approx |X_i|$.

Note that the converse of Corollary 1 is not true. For two nodes n_i, n_{i+1} for which $|X_i \bigcap X_{i+1}| \approx |X_i|$, we cannot conclude that both nodes are honest. Two colluding nodes may be incident to a link, and thus claim similar audit replies regardless of the packets forwarded.

Adaptive Audits with Cut Questions. We now show how the source can identify misbehaving nodes using an adaptive strategy and cut questions. Cut questions can be implemented by auditing one node at a time. These questions are of the form, "Is the misbehaving node upstream of n_i?", where n_i is the audited node. Assume there exists a single continuously misbehaving node n_M in P_{SD}. Define the set of nodes suspicious of misbehavior as $\mathcal{V} = \{n_1, \ldots, n_k\}$. If $n_i \in \mathcal{V}$ is audited and replies with X_i such that $|X_S \bigcap X_i| \ll |X_S|$, the source concludes that all nodes downstream of n_i are behaving honestly, and therefore $n_M \leq n_i$. This is true since either n_i is honest in which case it never received packets in X_S indicating an upstream misbehaving node, or n_i is the misbehaving node lying about its audit reply. If n_i replies that $|X_S \bigcap X_i| \approx |X_S|$, the source concludes that all nodes upstream of n_i are honest, and therefore $n_M \geq n_i$. This is true, since if any node upstream of n_i was the misbehaving one, n_i would not have received packets in X_S. Thus the set $\mathcal{V}$ is reduced to $\{n_i, \ldots, n_k\}$.

Pelc [25] proposed a questioning strategy for adaptive games in which the questioner wins if he determines ω, or proves a lie took place. For a search space of size $|\Omega|$, and a maximum number of lies ℓ, the winning strategy requires $\lceil \log_2 |\Omega| \rceil + \ell$ questions. To find ω, the questioner first performs a binary search requiring $\lceil \log_2 |\Omega| \rceil$ questions to converge to a value ω'. It then asks the responder

ℓ times if $\omega \leq \omega'$. Since the responder is limited in lies, the questioner can determine if ω' is the secret value or the responder has lied.

Following the winning strategy proposed by Pelc, let the source win if either a misbehaving link is identified or the source can prove a lie has occurred. The source can converge to a single link by performing a binary search. The source initializes $\mathcal{V} = \{n_1, \ldots, n_k\}$ and selects node with index $i = \lceil \frac{|\mathcal{V}|}{2} \rceil$, for audit. As previously described, $\mathcal{V}$ is reduced to either $\{n_1, \ldots, n_i\}$ or $\{n_i, \ldots, n_k\}$. The source continues to audit nodes in $\mathcal{V}$ until $|\mathcal{V}| = 2$. In the case of a single misbehaving node, the source identifies the misbehaving link as shown in the following Proposition.

Proposition 2. *For a single misbehaving node, the source always converges to the misbehaving link in* $\log_2(|P_{SD}|)$ *audits.*

Proof. Let n_M denote the misbehaving node. Initially, $\mathcal{V} = P_{SD}$ and hence $n_M \in \mathcal{V}$. Let the source select a node n_i upstream of n_M for audit. Being upstream, n_i responds honestly that it forwarded packets to the next hop, reducing $\mathcal{V}$ to $\{n_i, \ldots, n_k\}$, with $n_M \in \mathcal{V}$. Similarly, if a node n_j downstream of n_M is audited, it will respond that no packets were forwarded, reducing $\mathcal{V}$ to $\{n_1, \ldots, n_j\}$. If n_M is audited, its response will indicate that misbehavior occurs either upstream of downstream. In either case $n_M \in \mathcal{V}$, since the audited node always remains in $\mathcal{V}$. The convergence of the binary search will end in a suspicious set $\mathcal{V} = \{n_{M-1}, n_M\}$ or $\mathcal{V} = \{n_M, n_{M+1}\}$, depending on whether n_M indicated that misbehavior occurs upstream of downstream. In any case, the identified link is a misbehaving one since per the definition, its two incident nodes provide conflicting audit replies. Since the binary search converges in $\log_2(|P_{SD}|)$, in case $|M| = 1$ the source will locate n_M in $\log_2(|P_{SD}|)$ steps.

If two or more nodes collude, the source may converge on a link in which both nodes are behaving, as shown in the following example. In Figure 2(a), $M = \{n_1, n_4\}$ with nodes n_1, n_4 colluding. Initially, n_4 drops all packets, while n_1 behaves. Let node n_2 be audited and report no misbehavior, thus $\mathcal{V} = \{n_2, n_3, n_4\}$. Assume now that nodes n_1, n_4 switch their behavior with node n_1 dropping packets while n_4 is behaving, as shown in Figure 2(b). If node n_3 is audited, it will report misbehavior upstream, reducing $\mathcal{V}$ to $\{n_2, n_3\}$ and thus removing n_4 from $\mathcal{V}$. Hence, link (n_2, n_3) is incorrectly identified as misbehaving.

Pelc solves this problem through the repetitive questioning of the result, thereby exhausting the responder's lies. In our case, a simultaneous audit on nodes n_i, n_{i+1} of an identified link $\mathcal{V} = \{n_i, n_{i+1}\}$ is sufficient to identify a misbehaving link or the occurrence of a lie. If $|X_i \bigcap X_{i+1}| \ll |X_i|$, a misbehaving link is identified. Else, the source concludes that a lie occurred. Returning to our previous example, in Figure 2(c), n_2 and n_3 are simultaneously audited. Since both nodes are honest, they return identical audit replies and no misbehaving link is identified. In this example, the responder has lied by changing the value of ω during the search, i.e., initially $\omega = n_4$, then $\omega = n_1$. However, S can identify that a lie occurred.

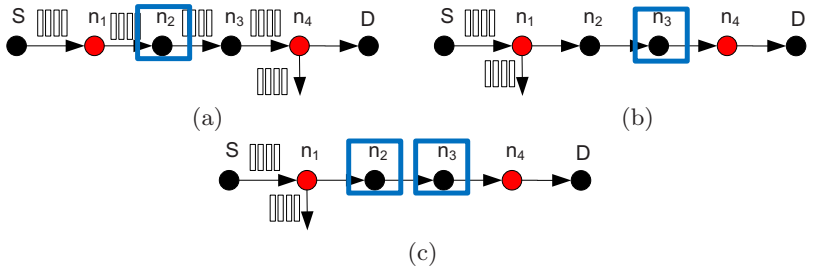

Fig. 2. (a) Nodes n_1, n_4 collude, with n_4 dropping all packets. Audited node n_2 claims misbehavior is downstream. (b) Nodes n_1, n_4 alter their behaviors, with n_1 dropping all packets. Audited node n_3 claims misbehavior is upstream. (c) Source simultaneously audits n_2, n_3 to verify if misbehaving link exists.

When the source identifies a lie occurred, is can also reach to the following conclusion: either (a) $n_M \in \mathcal{V}$ but lied during the simultaneous audit, or (b) $|M| \geq 2$ with at least one misbehaving node upstream of n_{i+1} and one downstream of n_i. Note that if $|M| = 1$ and the misbehaving node stops misbehaving (due to the fact that it is being audited) the destination alerts the source that misbehavior has stopped in P_{SD}. In such a case, the source will take two steps. First, any outstanding audits will be discarded. Second, the search will be suspended at the current state until misbehavior re-appears on P_{SD}. When misbehavior is resumed, the source continues the search from where it left off the last time misbehavior occurred.

If the destination does not alert the source that performance in P_{SD} has been restored, the source concludes that $|M| \geq 2$. This is evident in our example by the responses of n_2; on the first audit in Figure 2(a), it claims that misbehavior is downstream, while in Figure 2(c), it claims misbehavior is upstream. Let the audit process converge to link (n_i, n_{i+1}). Since the source knows that at least one misbehaving node is upstream of n_i and one is downstream, it attempts to isolate the effect of the misbehavior of each node by partitioning P_{SD} into $P_{Sn_i} = \{n_1, \ldots, n_i\}$ and $P_{n_{i+1}D} = \{n_{i+1}, \ldots, n_k\}$. The source repeats the audits recursively for each path partition $P_{Sn_i}, P_{n_{i+1}D}$. However, note that the destination can only determine if misbehavior occurs in P_{SD}, not which partition.

To treat each partition individually, the source considers n_i as a pseudo-destination and n_{i+1} as a pseudo-source. In P_{Sn_i}, node n_i is always audited simultaneously with any other node. Similarly node n_{i+1} is audited simultaneously with any other node in P_{n+1D}. Note that if n_i is the misbehaving node, it has only two strategies, (a) respond honestly, or (b) lie. If n_i lies, it immediately implicates itself in a misbehaving link, since both n_i, n_{i+1} are always audited. If n_i responds honestly, the search in P_{Sn_i} will converge to the misbehaving link (assuming one misbehaving node in P_{Sn_i}). For the realization of the cut questions, the source initializes $\mathcal{V}_{Sn_i} = \{n_1, \ldots, n_i\}$ and selects $n_j, j = \lceil \frac{|\mathcal{V}_{Sn_i}|}{2} \rceil$ for audit. The cut question "Is $n_M < n_j$?" is true if $|X_S \bigcap X_j| \ll |X_S|$ and $|X_S \bigcap X_i| \ll |X_S|$. The second condition verifies misbehavior on P_{Sn_i}.

Algorithm 1. Cut Questioning Algorithm

$n_i \leftarrow n_1, n_j \leftarrow n_{|P_{SD}|}, \mathcal{V} = \{n_i, \ldots, n_j\}$
while $|\mathcal{V}| > 2$ **do**
  $h = \lceil \frac{|\mathcal{V}|}{2} \rceil$, $Audit(n_h)$
  if $|X_S \bigcap X_h| \approx |X_S|$ **then**
    $n_i \leftarrow n_h$
  else
    $n_j \leftarrow n_h$
  end if
end while
$Audit(n_i, n_j)$
if $|X_i \bigcap X_j| \ll |X_i|$ **then**
  return X_i, X_j
else
  return $|M| \geq 2, Partition\ P_{SD}$
end if

Likewise on $P_{n_{i+1}D}$, the audit response of n_{i+1} acts as a verification if packets from X_S have reached this partition. Node n_{i+1} therefore acts as a pseudo-source for $P_{n_{i+1}D}$. Much like n_i, if n_{i+1} lies it immediately implicates itself in a misbehaving link since (n_i, n_{i+1}) is always audited. Thus the source can identify multiple misbehaving links using this adaptive auditing strategy. This strategy is presented in Algorithm 1.

Adaptive Audits with Membership Questions. Our scheme can also use an adaptive auditing strategy based on membership questions to identify the misbehaving nodes. Membership questions are constructed by combining two cut questions. To answer the question, "Is $n_M \in A = \{n_i, \ldots, n_j\}$?" the source audits n_i, n_j simultaneously and compares their audit replies. If $|X_i \bigcap X_j| \approx |X_i|$, then n_i, n_j claim $n_M \notin A$, since all packets forwarded by n_i are received by n_j. Else, they claim $n_M \in A$. Dhagat et al. [10] proposed an adaptive questioning strategy which proceeds in stages. During each stage, the questioner either believes the responder's answer and places it in a trusted set T, or discards it if it contradicts prior answers. Let $\mathcal{V}_j$ represent the set of possible values for ω at stage j, with $\mathcal{V}_1$ initialized to Ω.

Suppose that $\mathcal{V}_j$ is the current stage, with $|\mathcal{V}_j| > 1$, and let set $\{r_{j-1,a}, r_{j-1,b}\}$ represent the answers to round $j-1$. The questioner divides $\mathcal{V}_j$ into two equal-sized subsets, A and B. The responder is asked "Is $\omega \in A$?" If the answer $r_{j,a}$ is "yes", the questioner adds $\{r_{j,a}\}$ to T and moves to the next stage with $\mathcal{V}_{j+1} = A$. Else, the questioner asks "Is $\omega \in B$?" If the answer $r_{j,b}$ is "yes," $\{r_{j,a}, r_{j,b}\}$ are added to T and the questioner moves to $\mathcal{V}_{j+1} = B$. If both $r_{j,a}, r_{j,b}$ are negative, the questioner removes $\{r_{j-1,a}, r_{j-1,b}\}$ from T, and returns to stage $\mathcal{V}_{j-1}$. The questioner then selects a different partition of $\mathcal{V}_{j-1}$ for stage j and repeats the questioning on each partition. Dhagat et. al. showed that the responder's secret value ω can be identified after $q = \lceil \frac{2 \log_2 |\Omega|}{1-3\beta} \rceil$ questions, when $\beta < \frac{1}{3}$, with β being the fraction of q than are lies [10]. To prevent repeated lies

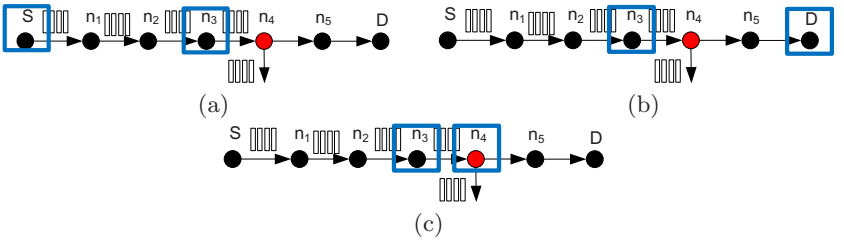

Fig. 3. (a) Let $\mathcal{V}_1 = \{S, n_1, \ldots, n_5, D\}$ with $A = \{S, n_1, n_2, n_3\}, B = \{n_3, n_4, n_5, D\}$ and $n_M = n_4$. The source audits A, concluding $n_M \notin A$. (b) The source then audits B, concluding $n_M \in B$. (c) The source proceeds to stage $\mathcal{V}_2 = \{n_3, n_4, n_5, D\}$ and continues the auditing strategy.

from the same misbehaving node, the source selects a new node and repeats the membership questions, until $|\mathcal{V}_j| = 2$.

Mapping Dhagat's questioning strategy to misbehavior identification, the source begins from stage $\mathcal{V}_1 = \{S, n_1, \ldots, n_k, D\}$. Set $\mathcal{V}_1$ is divided into two subsets, $A = \{S, \ldots, n_i\}$ and $B = \{n_i, \ldots, D\}$ with $i = \lceil \frac{|\mathcal{V}_1|}{2} \rceil$. The source first asks "Is $n_M \in A$?" by simultaneously auditing nodes S, n_i. If S and n_i return conflicting audit replies, the source knows that $n_M \in A$, adds $\{r_{1,a}\}$ to T, and proceeds to stage $\mathcal{V}_2 = \{S, \ldots, n_i\}$. Else, the source questions "Is $n_M \in B$?" by simultaneously auditing nodes n_i, D, whose audit replies define answer $r_{1,b}$. If n_i, D return conflicting audit replies, i.e., $|X_i \bigcap X_D| \ll |X_i|$, the source knows that $n_M \in B$, adds $\{r_{1,a}, r_{1,b}\}$ to T, and proceeds with $\mathcal{V}_2 = \{n_i, \ldots, D\}$. If both $r_{1,a}, r_{1,b}$ are negative, the source concludes a lie has occurred.

In Figure 3(a), $n_4 = n_M$. The source splits $\mathcal{V}_1 = \{S, n_1, \ldots, n_5, D\}$ to sets $A = \{S, n_1, n_2, n_3\}, B = \{n_3, n_4, n_5, D\}$, and audits S, n_3 to realize the membership question "Is $n_M \in A$?" Since n_3 is honest, the source asks "Is $n_M \in B$?" by simultaneously auditing n_3, D, as shown in Figure 3(b). Since n_3, D are honest, the source concludes $n_M \in B$. In Figure 3(c), the source moves to the next stage by dividing $\mathcal{V}_2 = B$ into two memberships sets. The process is repeated until $|\mathcal{V}_j| = 2$. In our example, the source converges to the misbehaving link (n_3, n_4). The source's auditing strategy is presented in Algorithm 2.

Proposition 3. *For a single misbehaving node, the source converges to the misbehaving link in less than* $4\log_2(|P_{SD}|) + 2$ *audits.*

Proof. Let the source be at stage $\mathcal{V}_j = \{n_i, \ldots, n_k\}$ with $n_M \in \mathcal{V}_j$ and select node n_h for audit, creating membership sets $A = \{n_i, \ldots, n_h\}$ and $B = \{n_h, \ldots, n_k\}$. If $n_M \neq n_i, n_h, n_k$, then all audit responses will be honest and the source will conclude either $n_M \in A$ or $n_M \in B$, thus proceeding to the next stage with $\mathcal{V}_{j+1} = A$, $\mathcal{V}_{j+1} = B$ and $n_M \in \mathcal{V}_{j+1}$. As long as the source audits honest nodes, the set of suspicious nodes V_j will be reduced by half.

Now assume one of the n_i, n_h, n_k is n_M. When audited, n_M will either respond honestly, or lie. If n_M responds honestly, the search will proceed to state $\mathcal{V}_{j+1}$

Algorithm 2. Membership Questioning Algorithm

$\mathcal{V}_1 = \{n_i, \ldots, n_k\}, n_i \leftarrow S, n_k \leftarrow D, T = r_{1,a}$
while $|\mathcal{V}_j| > 2$ **do**
  $h = \lceil \frac{|\mathcal{V}_j|}{2} \rceil, \quad r_{j,a} = audit(n_i, n_h)$
  if $|X_i \bigcap X_h| \ll |X_i|$ **then**
    $T \leftarrow \{r_{j,a}\}, \quad j = j + 1, \quad \mathcal{V}_j = \{n_i, \ldots, n_h\}$
  else
    $r_{j,b} = audit(n_h, n_k)$
    if $|X_h \bigcap X_k| \ll |X_h|$ **then**
      $T \leftarrow \{r_{j,a}, r_{j,b}\}, \quad j = j + 1, \quad \mathcal{V}_j = \{n_h, \ldots, n_k\}$
    else
      return $j = j - 1$
    end if
  end if
end while
return X_i, X_k

with $n_M \in \mathcal{V}_{j+1}$ and $|\mathcal{V}_{j+1}| = \frac{|\mathcal{V}_j|}{2}$. Thus the search continues to converge. If n_M lies, the source will obtain negative answers from both membership questions, unable to reduce $\mathcal{V}_j$ further, thus returning to stage $\mathcal{V}_{j-1}$ with $n_M \in \mathcal{V}_{j-1}$. The source will then pick a different n_h, and repeat the set splitting, thus preventing the same lie from repeating.

In the absence of lies, the total number of membership questions needed for convergence to the misbehaving link is $2\log_2(|P_{SD}|)$. This is true, since at each stage we split the suspicious set in half similar to a binary search. To realize a membership question we need to simultaneously audit two nodes, requiring a total of $4\log_2(|P_{SD}|)$ audits in the worst case. If n_M is audited and lies, the search backtracks to the previous stage, resulting in the waste of two audits. For a single misbehaving node n_M and the fact that the source always selects a different node after a backtrack, n_M will be audited only once. Thus, in the worst case, the source requires $q \leq 4\log_2(|P_{SD}|) + 2$ audits.

Corollary 2. *The source never converges to a link with two behaving nodes.*

Proof. According to Algorithm 2, the source must receive conflicting reports from two simultaneously audited nodes to proceed from stage $j-1$ to stage j. Hence, to terminate with $\mathcal{V}_j = \{n_i, n_{i+1}\}$ the source must receive conflicting audit replies from n_i, n_{i+1} when simultaneously audited. However, via Corollary 1, this cannot occur if n_i, n_{i+1} are behaving nodes.

It is possible that multiple neighboring colluding nodes can delay the search indefinitely. Assume all nodes in $\mathcal{V}_j$ collude. Once in stage $\mathcal{V}_{j+1}$, the replies to the audits from the colluding nodes yield membership questions on both partitions negative, thus forcing the source to return to stage $\mathcal{V}_j$. Auditing any other node in $\mathcal{V}_j$ will yield the same results since nodes in $\mathcal{V}_j$ are colluding. If the source has audited all possible partitions of $\mathcal{V}_j$, and thus all $n_i \in \mathcal{V}$, with no progress to the next stage, it terminates the search and proceeds to the identification phase.

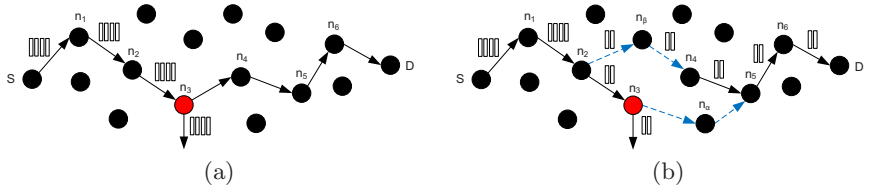

Fig. 4. (a) Node n_3 drops packets, with link (n_3, n_4) being the misbehaving link. (b) Slight alteration to routing path.

4.3 Misbehaving Node Identification

Once the source has converged to a misbehaving link (n_i, n_{i+1}), it can no longer proceed to identify the misbehaving node. The two conflicting audit responses from n_i, n_{i+1} indicate that either n_i or n_{i+1} is lying. From the routing point of view, identifying the misbehaving link is sufficient for restoring the performance in P_{SD} since the source can now avoid this link. However, we would like to identify and isolate the misbehaving node to prevent it from further affecting other paths. This is accomplished through the idea of path division. The path P_{SD} is divided in such a way that new independent observations can be made with respect to n_i and n_{i+1}. We first illustrate the idea of path division for a single misbehaving nodes and then generalize to many.

Single Misbehaving Node. Without loss of generality assume that the audit process converged to (n_M, n_{M+1}), where n_M is the misbehaving node. The source divides P_{SD} into two paths such that packets are routed through either n_M or n_{M+1}, and attempts to re-identify the misbehaving link. This can be achieved by bypassing each node in P_{SD} via an alternative path. Instead of performing the entire audit process, the source concentrates on the nodes around n_M, n_{M+1}, For example, in Figure 4(a), the source has identified link (n_3, n_4) as the misbehaving one. In Figure 4(b), the source splits the traffic between two paths that bypass n_3, n_4 in turn via nodes n_α, n_β. Path segment $\{n_2, n_3, n_4, n_5\}$ is replaced by the segments $\{n_2, n_\beta, n_4, n_5\}$ and $\{n_2, n_3, n_\alpha, n_5\}$, thus isolating n_3, n_4 from each other. The source simultaneously audits nodes n_β, n_4 and n_3, n_α to identify the misbehaving link. The source identifies link (n_3, n_α) as misbehaving, and hence identifies the misbehaving node n_3.

Multiple Misbehaving Nodes. Assume now the existence of multiple misbehaving nodes in P_{SD}. If the cut auditing strategy is employed, the source will split P_{SD} to smaller paths in order to isolate the effect of each misbehaving node. The source can then perform the path division in each subpath as in the case of a single misbehaving node. Note that, as in the case of a single misbehaving node, the newly added nodes must not be misbehaving in order to avoid framing honest nodes. If the membership questioning strategy is employed, the source will converge to a set $\mathcal{V}_j$ containing at most one honest node. To identify the misbehaving one, all nodes in $\mathcal{V}_j$ must be excluded in turn from P_{SD} according

to the path division process. That is, the source constructs $|\mathcal{V}_j|$ individual paths with each node in $\mathcal{V}_j$ being present on only one path.

4.4 Mobility

We now relax our assumption that P_{SD} does not change during the identification process. Let a node n_i be removed from P_{SD}. If $n_i \notin \mathcal{V}$, then its removal has no effect on the search. The source identifies misbehaving links from the nodes in $\mathcal{V}$. Let $n_i \in \mathcal{V}$. There are two cases, either n_i is a behaving node, or n_i is misbehaving. If n_i is behaving, then removing it is analogous to reducing $\mathcal{V}$ to a smaller set that still contains the misbehaving node. If n_i is misbehaving, then the performance in P_{SD} is restored or one less misbehaving node is present.

Consider now adding a new node n_i to P_{SD}. If n_i is added between nodes in $\mathcal{V}$, then regardless of n_i's behavior, this is equivalent to n_i being in $\mathcal{V}$, in the first place and not yet been audited. Let n_i be added in P_{SD} outside $\mathcal{V}$. If n_i is an honest node, there is no effect on the audit process. If n_i is a misbehaving node, then this is equivalent to the situation in which $|M| \geq 2$ and one of the n_M has been removed from $\mathcal{V}$. However, we have shown that both auditing strategies can address the case of multiple misbehaving nodes. In the case of cut questions, the source splits P_{SD} into two paths while in the case of membership questions, the source converges on the misbehaving node in $\mathcal{V}$. Once this node is removed, the source will continue to identify the newly added misbehaving node.

5 The Audit Mechanism

We now describe how the source can perform audits in a resource-efficient manner. The audit mechanism is adopted from [18] and is based on the compact representation of a membership set via Bloom filters [4]. The goal of auditing a node $n_i \in P_{SD}$ is to force n_i to commit to the set of packets X_i that it received and forwarded to the next hop. Contradicting commitments are used to identify misbehaving links and eventually misbehaving nodes. To respond to an audit, the node n_i records the packets forwarded for a period of time, and reports them to the source. Based on this report, the source compares the packets in X_i with the packets in X_S originally sent to the destination. Buffering the packets themselves requires a large amount of storage and significant overhead for transmission back to the source. On the other hand, Bloom filters provide a storage-efficient way of performing membership testing [4]. The audit process occurs in three steps; sending an audit request, constructing the audit reply, and processing the audit reply. We now describe these steps in detail.

Sending an Audit Request: The source audits a node n_i according to the algorithms described in Section 4. The source selects the audit duration a_d, measured in number of packets, and the initial packet sequence number a_s from which the audit will begin. The value of a_d is a parameter that must be sufficiently large to differentiate misbehavior from normal packet loss. The audit request is routed

to n_i via P_{SD}. Values a_s and a_d are randomized thereby preventing any misbehaving nodes from conjecturing the start and duration of audits, unless they are audited themselves. Note that an audit request may fail to reach the audited node n_i since a misbehaving node along P_{Sn_i} may drop it, or n_i is the misbehaving node and chooses not to respond. In this case, the source tries a threshold number of times to audit n_i. Failure to obtain a reply is interpreted as "Node n_i did not forward packets in X_S to the next hop." This is true since either n_i is the misbehaving node or a misbehaving node is upstream of n_i.

Constructing an Audit Reply: When a node n_i is audited, it constructs a Bloom filter of the set of packets it receives and forwards, from a_s to $a_s + a_d$, denoted by $X_i = \{x_{a_s}, x_{a_s+1}, \ldots, x_{a_d}\}$. By using a Bloom filter, packets in X_i can be compactly represented in an m-bit vector v_i with $m \ll |X_i|$ [4]. After a_d packets have been added to v_i, node n_i signs v_i, and sends it to S via the reverse path P_{n_iS}. The signed Bloom filter binds the audited node to the set of packets X_i that it claims to have forwarded to the next hop, in a publicly verifiable manner. Based on n_i's signature, any node can verify the authenticity and integrity of v_i. To assess the behavior of audited nodes, the source constructs its own Bloom filter v_S in the same manner as n_i. When S receives n_i's Bloom filter, it compares it against v_S and compute what fraction of packets in X_S was forwarded by n_i.

Processing the Audit Reply: When S receives v_i, it verifies its authenticity and discards v_i if the signature check fails. Otherwise, given the vector length m, the cardinalities of X_i, X_S, filters v_i, v_S, and the number z of hash functions used to generate the Bloom filters, S computes the metric [5],

$$|X_S \bigcap X_i| \approx |X_S| + |X_i| - \frac{\log_2\left(\frac{<v_S, v_i>}{m} + \left(1 - \frac{1}{m}\right)^{z|X_S|} + \left(1 - \frac{1}{m}\right)^{z|X_i|}\right)}{z \log_2\left(1 - \frac{1}{m}\right)} \quad (1)$$

6 Performance Evaluation

6.1 Simulation Setup

We randomly deployed 100 nodes within an 80×80 square and selected 10 source/destination pairs. For each pair, we constructed the shortest path and randomly selected the set of misbehaving nodes. We generated traffic from S to D according to the constant bit-rate (CBR) model. Each misbehaving node randomly selected a behavioral state of either *behave* or *misbehave*, with equal probability. It then randomly selected the duration of the state from the interval $[1, 400]$ packets. We focus on two metrics of interest: (a) the *communication overhead* defined as the number of messages transmitted/received by nodes in P_{SD}, weighed by 1/0.5, respectively [12], and (b) the *identification delay* defined as the time elapsed from the occurrence of misbehavior until the misbehaving nodes are identified, normalized over the audit duration. Simulations were performed in a packet-level C simulator.

6.2 Auditing Strategy Comparison

We first compared the performance of the two auditing strategies; the strategy based on cut questions as described by Algorithm 1, which we will refer to as CUT, and the strategy based on membership questions as described by Algorithm 2, which we will refer to as MEM.

Communication Overhead. In Figure 5(a), we show the communication overhead required to identify one misbehaving node as a function of the path length. We observe that CUT requires less communication overhead than MEM. This is expected, as the realization of cut questions requires only one audit, whereas membership questions require two audits. Both auditing strategies audit in a binary fashion, thus resulting in logarithmic increase in communication overhead as a function of the path length. In Figure 5(b), we show the communication overhead required to identify two misbehaving nodes as a function of path length.

Identification Delay. In Figure 5(c), we show the delay required to identify one misbehaving node as a function of the path length. Both CUT and MEM incur approximately the same delay due to their binary search approach. In Figure 5(d), we show the delay required to identify two misbehaving nodes as a

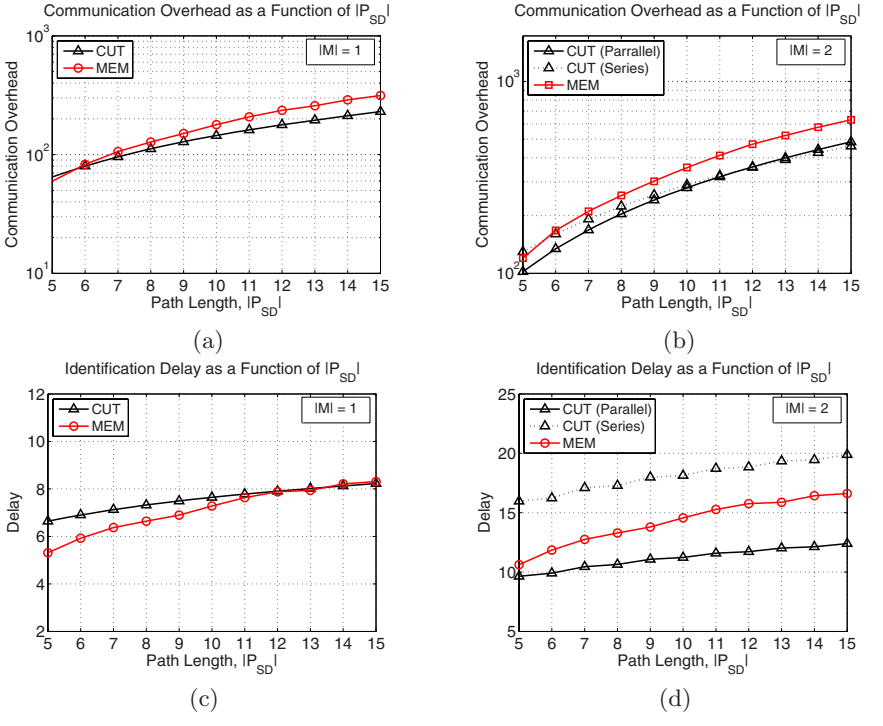

Fig. 5. Communication overhead for (a) one misbehaving node, (b) two misbehaving nodes. Identification delay for (c) one misbehaving node, (d) two misbehaving nodes.

function of the path length. In CUT, after the path is partitioned, the auditing of the two partitions is dependent on the misbehavior strategies of nodes in M. Assume that only one misbehaving node drops packets at a time. Thus the search will only audit the path partition which is reporting misbehavior. This causes the source to search the partitions in series, i.e., one at a time. If both misbehaving nodes drop packets, the source can audit the two path partitions in parallel, since each path partition contains a source (or pseudo-source) and a destination (or pseudo-destination). This parallel auditing decreases the delay.

For CUT, we plot both the case of search in series and parallel, giving an expected range for the delay. Note that the delay of MEM falls within this range; closer to the parallel CUT for smaller path sizes and closer to the series CUT as the path length increases. This is due to the nature of the auditing strategies employed. In CUT, the source cannot determine if a lie occurred until performing the simultaneous audit at the end of the auditing strategy. In MEM, the source determines if a lie occurred by looking for contradictions at every stage. Therefore, if a lie is found, the penalty is only the waste of two audits. This results in a tradeoff in which MEM incurs an additional overhead per stage compared to CUT by checking for contradictions at the expense of delay.

6.3 Comparison with Other Schemes

We now compare the performance of our algorithms to CONFIDANT [6], 2ACK [20], and AWERBUCH [1]. For CONFIDANT, every one-hop neighbor of a transmitting node was assumed to operate in promiscuous mode, thus overhearing transmitted messages. For 2ACK, a fraction p of the messages transmitted by each node was acknowledged two hops upstream of the receiving node. We set $p = \{1, 0.5, 0.1\}$ [20]. AWERBUCH identifies misbehaving links by requesting selected nodes in P_{SD} to acknowledge each packet back to the source. For comparison, we select the adaptive auditing strategy utilizing cut questions. The plots of Figure 5(a)-(d) can be used for comparisons with MEM. We first considered the overhead during a fixed duration of time, i.e., the time required to identify the misbehaving node using CUT.

Fixed Time Communication Overhead. In Figure 6(a), we show the communication overhead as a function of the path length. The Y axis is shown in logarithmic scale. The communication overhead for CUT is between 1-2 orders of magnitude less compared to other schemes. This gain is due to the fact that CUT does not expend energy on a per-packet basis to monitor the behavior of each node. The 2ACK scheme presents the highest communication overhead since every packet requires a 2-hop acknowledgment upstream per link traversed.

In Figure 6(b), we show the communication overhead as a function of the audit duration a_d for a path of eight nodes. Schemes 2ACK, CONFIDANT, and AWERBUCH all incur a linear increase in communication overhead, due to the per-packet behavior evaluation. On the other hand, the communication overhead for CUT and MEM is incurred on a per-audit basis, and is independent of audit duration. While our algorithms provide significant savings in communication

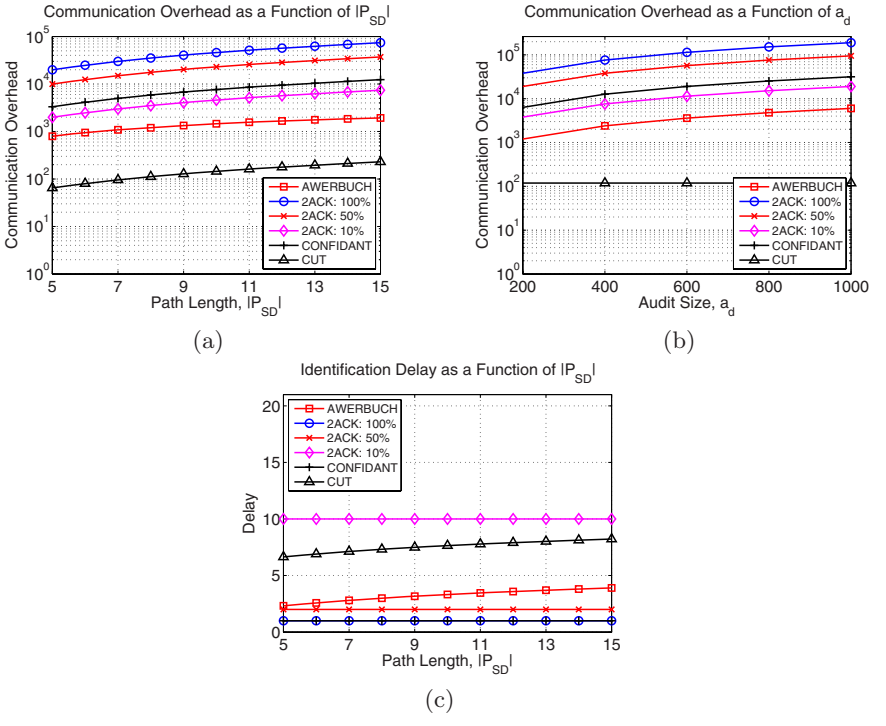

Fig. 6. (a) Communication overhead as a function of $|P_{SD}|$ for an audit size of 200 packets. The overhead is computed over time required by the CUT scheme to converge to the misbehaving node. (b) Communication overhead as a function of audit size for $|P_{SD}| = 8$. (c) Delay as a function of $|P_{SD}|$ in units of number of audits.

overhead, they require a longer time to identify the misbehaving nodes. On the other hand, the proactive schemes require only the duration of one audit to identify misbehavior. This is due to the fact that proactive protocols monitor all nodes in the path P_{SD} in parallel. Fortunately, for schemes CUT and MEM, the delay grows logarithmically with $|P_{SD}|$. Hence, the increase in identification delay is small compared to the savings in communication overhead.

In Figure 7(c), we show the identification delay as a function of path length. CONFIDANT requires a single audit duration to identify the misbehaving node since all nodes in P_{SD} are monitored in parallel. AWERBUCH performs a binary search, incurring a logarithmic increase in delay. The 2ACK scheme also requires a single audit duration for identification when all packets are acknowledged. However, the identification delay increases when only a fraction of the packets are acknowledged. For example, when only 10% of the packets are acknowledged, 2ACK and CUT incurr similar delay. However, as shown in Figure 6(b), CUT incurs an order of magnitude less in communication overhead.

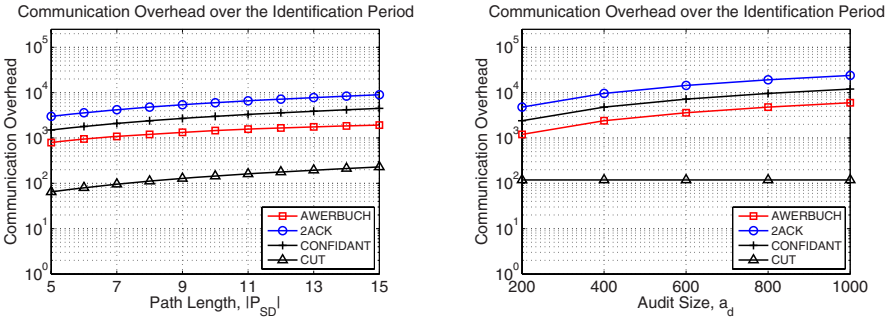

Fig. 7. (a) Communication overhead for an audit size of 200 packets. For each scheme, the overhead is computed for the time required to identify misbehavior, (b) communication overhead as a function of audit size for $|P_{SD}| = 8$.

Comparison Based on Identification Delay. We now evaluate the communication overhead incurred by each scheme until the misbehaving node is identified. In Figure 7(a), we show the communication overhead as a function of the path length, for an audit size of 200 packets. In Figure 7(b), we show the communication overhead as a function of the audit size, for a path of eight nodes. We observe that even in the case where the communication overhead is measured only during the identification delay, CUT significantly outperforms the other schemes. The CONFIDANT, 2ACK and AWERBUCH schemes are sensitive to path length and audit size. On the other hand, CUT illustrates a graceful tradeoff between communication overhead and delay.

7 Conclusion

We addressed the problem of identifying misbehaving nodes that refuse to forward packets to the destination in a wireless multi-hop network. We mapped this problem to the classic Rényi-Ulam game of 20 questions. From this mapping we employed communication efficient questioning strategies which allow the source to locate the set of misbehaving nodes. We showed that our scheme significantly reduces the communication overhead associated with misbehavior identification compared to previously proposed schemes. This reduction in resource expenditure comes at the expense of a logarithmic increase in the identification delay.

References

1. Awerbuch, B., Holmer, D., Rotaru, C.-N., Rubens, H.: An on-demand secure routing protocol resilient to byzantine failures. In: WiSe 2002 (2002)
2. Balakrishnan, K., Deng, J., Varshney, P.K.: Twoack: Preventing selfishness in mobile ad hoc networks. In: WCNC 2005 (2005)

3. Berlekamp, E.: Error Correcting Codes. Wiley, NY (1968)
4. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
5. Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. Internet Mathematics 1(4), 485–509 (2004)
6. Buchegger, S., Boudec, J.-Y.L.: Performance analysis of the confidant protocol (cooperation of nodes: Fairness in dynamic ad-hoc networks). In: MobiHOC 2002 (2002)
7. Buchegger, S., Boudec, J.-Y.L.: Self-policing mobile ad-hoc networks by reputation systems. IEEE Communications Magazine, 101–107 (2005)
8. Buttyan, L., Hubaux, J.-P.: Enforcing service availability in mobile ad-hoc wans. In: MobiHOC 2000, pp. 87–96 (2000)
9. Buttyan, L., Hubaux, J.-P.: Stimulating cooperation in self-organizing mobile ad hoc networks. ACM/Kluwer Mobile Networks and Applications 8(5) (2003)
10. Dhagat, A., Gács, P., Winkler, P.: On playing "twenty questions" with a liar. In: SODA 1992, pp. 16–22. Society for Industrial and Applied Mathematics (1992)
11. Dong, Y., Go, H., Sui, A., Li, V., Hui, L., Yiu, S.: Providing Distributed Certificate Authority Service in Mobile Ad Hoc Networks. In: SecureComm 2005 (2005)
12. Feeney, L.M., Nilsson, M.: Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In: INFOCOM 2001 (2001)
13. Ganeriwal, S., Srivastava, M.: Reputation-based framework for high integrity sensor networks. In: SASN 2004, pp. 66–77 (2004)
14. He, Q., Wu, D., Khosla, P.: Sori: A secure and objective reputation-based incentive scheme for ad hoc networks. In: WCNC 2004 (2004)
15. Hu, Y., Johnson, D., Perrig, A.: SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks. Ad Hoc Networks 1(1), 175–192 (2003)
16. Jakobsson, M., Hubaux, J.-P., Buttyan, L.: A micropayment scheme encouraging collaboration in multi-hop cellular networks. In: Proc. of Financial Crypto (2003)
17. Johnson, D., Maltz, D., Hu, Y.-C.: The dynamic source routing protocol for mobile ad hoc networks (dsr). draft-ietf-manet-dsr-09.txt (2003)
18. Kozma Jr., W., Lazos, L.: REAct: Resource-Efficient Accountability for Node Misbehavior in Ad Hoc Networks based on Random Audits. In: WiSec 2009 (2009)
19. Liu, A., Ning, P.: Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In: IPSN 2008 (2008)
20. Liu, K., Deng, J., Varshney, P., Balakrishnan, K.: An acknowledgment-based approach for the detection of routing misbehavior in manets. IEEE Transactions on Mobile Computing 6(5), 536–550 (2006)
21. Marti, S., Giuli, T., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: MobiCom 2000, pp. 255–265 (2000)
22. Michiardi, P., Molva, R.: Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In: CMS 2002 (2002)
23. Padmanabhan, V.-N., Simon, D.-R.: Secure traceroute to detect faulty or malicious routing. SIGCOMM Computer Communication Review 33(1) (2003)
24. Papadimitratos, P., Haas, Z.: Secure routing for mobile ad hoc networks. In: SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), pp. 1–27 (2002)
25. Pelc, A.: Detecting errors in searching games. Journal of Combinatorial Theory Series A 51(1), 43–54 (1989)
26. Perkins, C., Royer, E., Das, S.: Ad hoc On-Demand Distance Vector (AODV) Routing (2003)

27. Perrig, A., Szewczyk, R., Tygar, J., Wen, V., Culler, D.: SPINS: Security Protocols for Sensor Networks. Wireless Networks 8(5), 521–534 (2002)
28. Raghani, S., Toshniwal, D., Joshi, R.: Dynamic Support for Distributed Certification Authority in Mobile Ad Hoc Networks. In: Proceedings of the 2006 International Conference on Hybrid Information Technology, vol. 1, pp. 424–432. IEEE Computer Society, Washington (2006)
29. Rényi, A.: A Diary on Information Theory. Wiley, New York (1984)
30. Rivest, R., Meyer, A., Kleitman, D., Winklmann, K., Spencer, J.: Coping with errors in binary search procedures. J. Comput. System Sci. 20, 396–404 (1980)
31. Spencer, J., Winkler, P.: Three thresholds for a liar. Combinatorics, Probability and Computing 1, 81–93 (1992)
32. Ulam, S.: Adventures of a Mathematician. Scribner, New York (1976)
33. Yi, S., Kravets, R.: MOCA: Mobile Certificate Authority for Wireless Ad Hoc Networks. In: 2nd Annual PKI Research Workshop Pre-Proceedings, vol. 51
34. Zhong, S., Chen, J., Yang, Y.R.: Sprite: A simple cheat-proof, credit-based system for mobile ad-hoc networks. In: INFOCOM 2003 (2003)