AMD: Audit-based Misbehavior Detection in Wireless Ad Hoc Networks

Yu Zhang, Loukas Lazos, *Member, IEEE,* and William Jr. Kozma
Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, 85721
E-mail:{yuzhang, llazos}@ece.arizona.edu, kozmab@hotmail.com

We address the problem of identifying and isolating misbehaving nodes that refuse to forward packets in multi-hop ad hoc networks. We develop a comprehensive system called *Audit-based Misbehavior Detection* (AMD) that effectively and efficiently isolates both continuous and selective packet droppers. The AMD system integrates reputation management, trustworthy route discovery, and identification of misbehaving nodes based on behavioral audits. Compared to previous methods, AMD evaluates node behavior on a per-packet basis, without employing energy-expensive overhearing techniques or intensive acknowledgment schemes. Moreover, AMD can detect selective dropping attacks even if end-to-end traffic is encrypted and can be applied to multi-channel networks or networks consisting of nodes with directional antennas. We show via simulations that AMD successfully avoids misbehaving nodes, even when a large portion of the network refuses to forward packets.

Index Terms—Misbehavior, Packet dropping, Secure routing, Reputation System, Ad hoc networks, Wireless communications.

1 Introduction

In the absence of a supporting infrastructure, wireless ad hoc networks realize end-to-end communications in a cooperative manner. Nodes rely on the establishment of multi-hop routes to overcome the limitations of their finite communication range. In this paradigm, intermediate nodes are responsible for relaying packets from the source to the destination. As an example, consider Fig. 1 depicting a source S using a multi-hop path to route data to a destination D. This network model presupposes that intermediate nodes are willing to carry traffic other than their own. When ad hoc networks are deployed in hostile environments (tactical networks), or consist of nodes that belong to multiple independent entities, a protocol-compliant behavior cannot be assumed. Unattended devices can become compromised and drop transit traffic in order to degrade the network performance. Moreover, selfish users may misconfigure their devices to refuse forwarding traffic in order to conserve energy. This type of behavior is typically termed as *node misbehavior* [1], [19], [21].

Existing solutions for identifying misbehaving nodes either use some form of per-packet evaluation of peer behavior [5], [19]–[21], or provide cooperation incentives to stimulate participation [6], [33]. Incentive-based approaches do not address the case of malicious nodes who aim at disrupting the overall network operation. On the other hand, per-packet behavior evaluation techniques are based on either transmission overhearing [5], [20], [21] or issuance of per-packet acknowledgements [2], [19]. These monitoring operations must be repeated on every hop of a multi-hop route, thus leading to high communication overhead and energy expenditure. Moreover, they fail to detect dropping attacks of selective nature, since intermediate monitoring nodes may not be aware of the desired selective dropping pattern to be detected.

Preliminary versions of this paper were presented in WiSec 2009 as a short paper [18] and in SecureComm 2009 *[17].*
This research was supported in part by NSF (under grants CNS-0844111, CNS-1016943). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

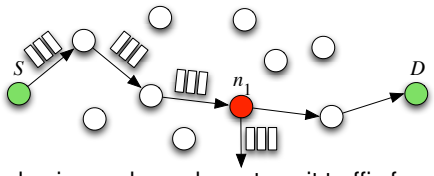

Fig. 1. Misbehaving node n_1 drops transit traffic from S to D.

Motivated by these limitations, we propose a system called *Audit-based Misbehavior Detection* (AMD), which achieves per-packet behavior evaluation without incurring a per-packet per-hop cost. AMD is a comprehensive solution that integrates identification of misbehaving nodes, reputation management, and trustworthy route discovery in a distributed and resource-efficient manner.

Our Contributions–We develop the AMD system for detecting and isolating misbehaving nodes. Compared to state-of-the-art, AMD provides the following additional features:

- AMD enables the per-packet evaluation of a node's behavior without incurring a per-packet overhead.
- AMD enables the concurrent first-hand evaluation of the behavior of several nodes that are not necessarily one-hop neighbors. Overhearing techniques are limited to one hop.
- AMD can operate in multi-channel networks and in networks with directional antennas. Current packet overhearing techniques are only applicable when transmissions can be overhead by peers operating on the same frequency band.
- AMD detects selective dropping behaviors by allowing the source to perform matching against any desired selective dropping patterns. This is particularly important when end-to-end traffic is encrypted. In the latter scenario, only the source and destination have access to the contents of the packets and can detect selective dropping.

We show that AMD can construct paths consisting of highly trusted nodes, subject to a desired path length constraint. When paths contain misbehaving nodes, these nodes are efficiently located by a behavioral audit process. We map the problem of identifying misbehaving nodes to the classic Rényi-Ulam game of 20 questions. The identification strategy is supplemented by

the knowledge of nodes' reputation. Audits are performed using storage-efficient membership structures [4].

Paper Organization–The remainder of the paper is organized as follows. In Section 2, we present related work. In Section 3, we state our network and adversary models and describe the AMD system architecture. In Sections 4, 5 and 6 we present the reputation, route discovery and audit modules, respectively. In Section 7, we evaluate the performance of AMD via extensive simulations. Section 8, summarizes our findings.

2 Related Work

Previously proposed methods for eliminating packet droppers can be classified into (a) credit-based systems [6], [7], [33], (b) reputation-based systems [5], [11], [13], [15], [21], [27], and (c) acknowledgment-based systems [2], [3], [19], [22], [23], [31].

Credit-Based Systems: Credit-based systems are designed to provide incentives for forwarding packets. Buttyan and Hubaux [6] proposed a system in which nodes accumulate credit for every packet they forward, and spend their credit to transmit their own packets. To ensure correctness, the credit counter is implemented in tamper-proof hardware. Zhong et al. [33] proposed Sprite, in which nodes collect *receipts* for the packets they forward to other nodes. When the node has a high-speed link to a Credit Clearance Service (CCS), it uploads its receipts and obtains credit. Crowcroft et al. [7] proposed a scheme that adjusts the credit reward to traffic and congestion conditions. While credit-based systems motivate selfish nodes to cooperate, they provide no incentive to malicious nodes. Such nodes have no intend to collect credit for forwarding their own traffic. Moreover, credit-based systems do not identify misbehaving nodes, thus allowing them to remain within the network indefinitely.

Reputation-Based Systems: Reputation-based systems use ratings for evaluating the trustworthiness of nodes in forwarding traffic. These ratings are dynamically adjusted based on the nodes' observed behavior. In the context of ad hoc networks, Ganeriwal and Srivastava [11] developed a Bayesian model to map binary ratings to reputation metrics, using a beta probability density function. Jøsang and Ismail [15] proposed a similar ranking system that utilized direct feedback received from one-hop neighbors. Michiardi and Molva [22] proposed the CORE mechanism for computing, distributing, and updating reputation values composed from disparate sources of information.

Reputation-based systems use neighboring monitoring techniques to evaluate the behavior of nodes. Marti et al. [21] proposed a scheme which relies on two modules, the *watchdog* and the *pathrater*. The watchdog module is responsible for overhearing the transmission of a successor node, thus verifying the successful packet forwarding to the next hop. The pathrater module uses the accusations generated by the watchdog module to select paths free of misbehaving nodes. Buchegger and Le Boudec [5] proposed a scheme called CONFIDANT, which extends the watchdog module to all one-hop neighbors that can monitor nearby transmissions (not just the predecessor node). When misbehavior is detected, monitoring nodes broadcast alarm messages in order to notify their peers of the detected misbehavior and adjust the corresponding reputation values. Similar monitoring techniques have also been used in [13], [27].

Transmission overhearing becomes highly complex in multi-channel networks or when nodes are equipped with directional antennas. Neighboring nodes may be engaged in parallel transmissions in orthogonal channels or different sectors thus being unable to monitor their peers. Moreover, operating radios in promiscuous mode for the purpose of overhearing requires up to 0.5 times the amount of energy for transmitting a message [10]. Note that for a multi-hop route, a given packet must be overheard by multiple monitors and over multiple hops, *thus making the monitoring operation more expensive than the actual end-to-end communication.* Finally, neighboring monitoring typically record coarse metrics of misbehavior such as packet counts that are inadequate to detect highly sophisticated selective dropping attacks tailored to specific applications. Motivated by the inadequacy and inefficiency of transmission overhearing, the monitoring approach developed in AMD incurs overhead on a per-flow basis instead of on a per-packet basis, thus having significantly smaller energy expenditure. Moreover, it allows the full customization of the misbehavior criteria for detecting a multitude of selective dropping strategies.

Acknowledgment-Based Systems: Acknowledgment-based systems rely on the reception of acknowledgments to verify that a message was forwarded to the next hop. Balakrishnan et al. [3] proposed a scheme called TWOACK, where nodes explicitly send 2-hop acknowledgment messages along the reverse path, verifying that the intermediate node faithfully forwarded packets. Packets that have not yet been acknowledged remain in a cache until they expire. A value is assigned to the quantity/frequency of un-verified packets to determine misbehavior.

Liu et al. [19] improved on TWOACK by proposing 2ACK. Similar to TWOACK, nodes explicitly send 2-hop acknowledgments to verify cooperation. Xue and Nahrstedt [31] proposed the Best-effort Fault-Tolerant Routing scheme, which relies on end-to-end acknowledgment messages to monitor packet delivery ratio and select routing paths which avoid misbehaving nodes. Awerbuch et al. [2] proposed an on-demand secure routing protocol (ODSBR) that identifies misbehaving links. The source probes intermediate nodes to acknowledge each packet and performs a binary search to identify the link where packets are dropped.

ACK-based systems also incur a high communication and energy overhead for behavioral monitoring. For each packet transmitted by the source, several acknowledgements must be transmitted and received over several hops. Moreover, they cannot detect attacks of selective nature over encrypted end-to-end flows.

3 System Model and Problem Statement

Network Model–We consider a multi-hop ad hoc network where any path P_{SD} 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 [14]. If DSR is not used, P_{SD} can be identified through a traceroute operation, or becomes known once the route is established. For simplicity, we number nodes in P_{SD} in ascending order, i.e., n_i is *upstream* of n_j if $i < j$. We further assume that the source can monitor the performance of P_{SD}. This can be achieved either in collaboration with the destination or implicitly at the transport layer (see Section 6).

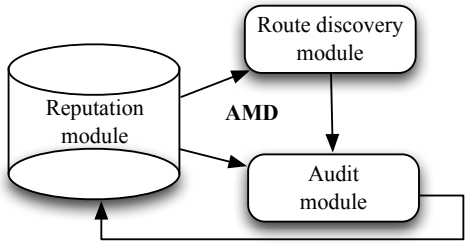

Fig. 2. The AMD system architecture.

To prevent user impersonation and message modification attacks, the integrity, authenticity, and freshness of critical control messages is verified using cryptographic methods. For example, a public key cryptosystem can be used to verify the authenticity and integrity of messages while providing confidentiality [28]. 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 (e.g., [9]). Moreover, new nodes can be added to the system after they are initialized by the CA. Every node n_i is assumed to be in possession of a private/public key pair denoted as (sk_i, pk_i). The public key is assumed to be known to all participating nodes. This is achieved either at initialization, or via the use of certificates. While any secure cryptosystem can be used for our purposes, in our work, we adopt a system that satisfies the homomorphic multiplication property such as RSA [28]. The development of a key management system for establishing trust within the network is beyond the scope of the present work.

Adversarial Model–A fraction of the nodes deployed in the network is assumed to be misbehaving. This misbehavior is manifested by the dropping of transit traffic from a source to a destination. Misbehaving nodes can be continuous droppers, or adopt any selective dropping strategy. For a given path P_{SD} of length k, we assume that a set of $\mathcal{M}$ misbehaving nodes, with $|\mathcal{M}| \leq k$, exist along this path. In our model, we do not consider other types of misbehavior against the routing process such as advertisement of false routing information, creation of sinkholes, blackholes, wormholes, etc. [16]. Moreover, we do not attempt to detect misbehaving nodes that are unwilling to participate in routing paths. Such nodes already isolate themselves from the network and do not actively degrade the network performance.

3.1 The AMD System Architecture

AMD provides a comprehensive misbehavior identification and node isolation system for eliminating misbehavior from a given network. This system consists of the integration of three modules: a *reputation* module, a *route discovery* module, and an *audit* module. These modules closely interact to coordinate the functions of misbehavior detection, discovery of trustworthy routes, and evaluation of the reputation of peers. A schematic of the relationship between the three modules of AMD is shown in Figure 2. The reputation module is responsible for managing reputation information based on the recommendations of the audit module. Reputation values are exploited by the route discovery module for establishing routes that exclude nodes with low reputations. Finally, the audit module efficiently identifies misbehaving nodes via an audit process. This process is accelerated based on input received from the reputation module. We note that while several techniques have been proposed for reputation management and reputation-based route discovery, in AMD, we develop novel methods for these two functions that integrate efficiently with the per-flow behavior evaluation implemented by the audit module. We now describe the three modules in detail.

4 The Reputation Module

AMD isolates misbehaving nodes by implementing a reputation-based system. Nodes with low reputation values are excluded from routing paths, thus being unable to drop transit traffic. The reputation module is responsible for computing and managing the reputation of nodes. We adopt a decentralized approach in which each node maintains its own view of the reputation of other nodes. Such implementation alleviates the communication overhead for transmitting information to a centralized location, and readily translates to the distributed nature of ad hoc networks. Moreover, it allows nodes to hold individualized reputation metrics for their peers depending on their direct and indirect interactions.

We take into account both *first-hand* and *second-hand* information. Information is considered to be first-hand if it is obtained by direct interaction between nodes (e.g., node n_i routes information via node n_j), and is considered to be second-hand if it is indirectly obtained based on the opinions of other nodes [22]. Consideration of both first-hand and second hand information has been shown to improve the reliability of reputation metrics [13], [21], [22].

4.1 Computation of Node Reputation Values

The reputation module running on a node n_i maintains a reputation vector $R_i(t) = \{r_i^1(t), r_i^2(t), \ldots, r_i^{\mathcal{N}}(t)\}$ that contains the reputation values $r_i^j(t)$ of all nodes $n_j \in \mathcal{N} \backslash \{n_i\}$. Here, $\mathcal{N}$ is the set of network nodes, and t denotes the current time epoch over which the reputation values are computed. Reputation values $r_i^j(t)$ are restricted to the range $(0, 1]$, are initialized to some value (e.g, 0.5), and are updated at the end of each epoch.

First-hand information: A reputation evaluation is considered to be first-hand, if it originates from the audit module running on n_i. This is because the audit module can make direct observations of the behavior of nodes in a path P_{SD} based on behavioral audits. To update the reputation based on first-hand information, we adopt an additive increase/multiplicative decrease (AIMD) algorithm [21], with a multiplicative factor $0 < \alpha < 1$ and an additive factor $0 < \beta < 1$. If during epoch t, the audit module of node n_i provided a first-hand evaluation of node n_j, the reputation value $r_i^j(t)$ is computed as,

$$r_i^j(t) = \begin{cases} \alpha \times r_i^j(t-1), & n_j \text{ misbehaves}, \\ \min\{r_i^j(t-1) + \beta, 1.0\}, & \text{otherwise}. \end{cases} \quad (1)$$

where the determination of n_j's behavior is provided by the audit module running on n_i, when $n_i = S$ and $n_j \in P_{SD}$. In (1), we have adopted the AIMD principle in order to rapidly isolate a misbehaving node from routing paths. Due to the multiplicative factor α, the reputation of a misbehaving node rapidly declines with repeated misbehavior. On the other hand,

a node with a low reputation would require significant time (in epochs) until its reputation is restored, due to the additive increase factor β. The difference between the reputation decrease and increase mechanism prevents a selectively misbehaving node from oscillating between periods of misbehavior and good behavior for the purpose of dropping packets while remaining in active paths. As it shown by our simulations in Section 7, such nodes are assigned low reputation values and are excluded from routing paths. Parameters α, β can be empirically tuned according to network characteristics such as number of alternative paths (network connectivity degree) and expected congestion conditions.

Second-hand information: Second-hand information is used only if first-hand information becomes stale, or is not available due to the lack of prior interaction between two nodes. First-hand information is considered stale if it has not been obtained for the last t_0 epochs. Disregarding stale reputation information implements an *aging* mechanism that prevents *sleeper attacks* wherein a node behaves for an initial period of time to gain high reputation, before exhibiting misbehavior [11]. In the absence of first-hand information, a node n_i averages all second-hand information reported by other nodes within the last t_0 epochs. Let $\mathcal{I}_i(t)$ denote the set of nodes that provided second-hand information to n_i within the last t_0 epochs. The reputation value is computed as,

$$r_i^j(t) = \frac{\sum_{m \in \mathcal{I}_i(t)} r_m^j(t)}{|\mathcal{I}_i(t)|}. \quad (2)$$

The reputation value in (2), is updated at every epoch by discarding all second-hand information older than t_0 epochs and admitting all new second-hand information. If no second-hand information is available for the last t_0 epochs, the reputation value is restored to the last known first-hand information. Also, if first-hand information becomes available, it replaces the second-hand information. To simplify our notations, we eliminate the parameter t from the reputation expressions, when unnecessary.

4.2 Collection of Reputation Information

When a source S establishes a route to a destination D, the audit module running on S makes evaluations on the behavior of each node along the path P_{SD}. These evaluations are considered as first-hand information for S. The source propagates first-hand information to all nodes along P_{SD}, thus providing second-hand information. More formally, consider a path $P_{SD} = \{S, n_1, n_2, \ldots, n_k, D\}$. Using the evaluations of the audit module, S computes $r_S^j, (j = 1 \ldots k)$ and updates its own reputation values using (1). It then distributes r_S^j to all nodes in $\{n_1, \ldots, n_k\} \backslash \{n_j\}$. For instance, node n_1 will receive all reputation values $r_S^j, (j = 2 \ldots k)$, except its own value r_S^1.

The reputation distribution operation implements a tradeoff between the communication cost of updating reputation values and obtaining a current view of the reputation of other nodes. Most prior methods are either limited to distribution within one hop (e.g, [1]), or employ some form of flooding. We adopt a path-oriented technique that achieves a desirable tradeoff between the two approaches. By limiting the propagation of second-hand information along P_{SD}, information is provided to nodes that are likely to participate in the same paths. Nodes located at disparate parts of the network do not often share reputation information.

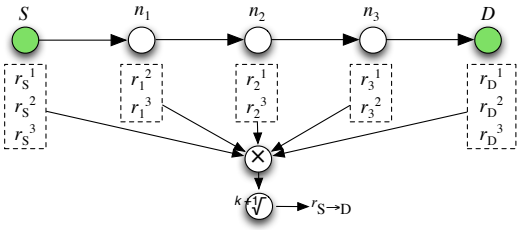

Fig. 3. Computing the path reputation value for P_{SD}.

Note that malicious nodes could distribute low reputation values for other nodes, in order to exclude them from the network (badmouthing). However, this attack is mitigated in our approach. Second-hand reputation is only used if first-hand reputation is not available. Moreover, the second-hand reputation value is computed via an averaging operation. A single false accusation would not be sufficient to exclude a falsely accused node from routing paths. Finally, even if a node is excluded from one path due to lies, this node could still be included in other paths, since every source holds its own reputation values. Assuming that the accused node is benign, sources in other paths provide positive feedback for the falsely accused node.

5 The Route Discovery Module

The route discovery module is responsible for the discovery of trustworthy paths from a source to a destination. This module is invoked by the source whenever there is no cached path to the destination. To quantify the trustworthiness of a path, we define the following path reputation metric.

Definition 1: **Path reputation value:** The reputation of a path $P_{SD} = \{S, n_1, n_2, \ldots, n_k, D\}$ is defined as:

$$r_{S \to D} = \sqrt[k+1]{\prod_{i=1}^{k} (r_S^i \cdot r_D^i \cdot \prod_{j=1, j \neq i}^{k} r_i^j)}. \quad (3)$$

The justification for using the metric in (3) is as follows: A path P_{SD} can be seen as an in-series system of independent components. The failure of one component (a node dropping packets) results in the failure of the entire system (path). The reliability of in-series systems is defined as the product of the reliability values of the individual components. Similarly, in (3), the trustworthiness of a path is defined as the product of the reputation values of the nodes that participate in that path. A subtlety in our definition is the fact that there is no universal reputation value for each node, but the reputation values are individual perceptions of trustworthiness of one node in regards to another. Hence, to compute the path reputation value, we multiply the reputation values r_i^j of intermediate nodes as perceived by all the nodes participating in P_{SD}. The calculation of $r_{S \to D}$ on a sample path is shown in Figure 3.

Here, we emphasize that none of the nodes in P_{SD} is providing an evaluation of itself. Hence, a malicious node with low reputation value cannot increase the path reputation to a value higher than its own reputation. A malicious node can, however, lower the reputation value of a path by lying about the reputation values of other nodes. This strategy decreases the path reputation, leading to the exclusion of the lying node from routing paths.

5.1 Discovery of Trustworthy Routes

To discover trustworthy routes, we modify the discovery phase of the DSR protocol [14]. In DSR, when S has packets for D, it checks whether a route exists in its cache. If a route does not exist, S broadcasts a Route Request (RREQ) message. This message contains the source ID, destination ID, and the time-to-live (TTL)[1]. Any intermediate node that receives the RREQ, appends its ID to the RREQ message and rebroadcasts it while decreasing the TTL field by one unit. If a receiving node is the destination, D responds to S with a route reply (RREP) message containing the entire P_{SD}. The RREP follows the reverse path to S. In AMD, RREQ and RREP messages include the *accumulated path reputation* (APR) field for calculating $r_{S\to D}$. Formally, the route discovery is as follows.

Step 1: S initializes the APR to a random $r_0 \in (0,1]$. It encrypts the APR with D's public key, and signs it. The broadcasted RREQ is as follows (where $||$ represents concatenation),

$$\text{RREQ: } S \;||\; D \;||\; \text{TTL} \;||\; E_{pk_D}(r_0) \;||\; E_{pk_D}(r_0) \;||\; sig_{sk_S}(r_0).$$

Step 2: Intermediate node n_i receiving a RREQ, updates the APR and TTL fields, appends its ID and rebroadcasts the RREQ.

$$\text{RREQ: } S \;||\; D \;||\; \text{TTL} \;||\; n_1,\ldots,n_i \;||\; E_{pk_D}(\text{APR}_i) \;||\; E_{pk_D}(r_0) \;||\; sig_{sk_S}(r_0),$$

where, $E_{pk_D}(\text{APR}_i) = E_{pk_D}(\text{APR}_{i-i}) \cdot E_{pk_D}(\prod_{j=1}^{i-1} r_i^j)$.

Step 3: For every RREQ indicating a unique path to the source, the destination decrypts the APR field using sk_D. It also recovers the initial random value r_0 and verifies the signature $sig_{sk_S}(r_0)$ using pk_S. D rejects the route if $\text{APR}_k \geq r_0$ (n_k is the last node before D). Otherwise, it obtains an estimate $\hat{r}_{S\to D}$ of $r_{S\to D}$:

$$\hat{r}_{S\to D} = \sqrt[\frac{k+1}{2}]{\frac{\text{APR}_k}{r_0}(\prod_{j=1}^{k} r_D^j)}.$$

For all paths of length within a factor $\lambda \geq 1$ of the shortest path (route selection factor) and with $\hat{r}_{S\to D} \geq \gamma_1$, $\gamma_1 \in (0,1]$, the destination issues a RREP message. The RREP contains an APR field which is now encrypted with pk_S. It also contains a second copy of $E_{pk_S}(\text{APR}_D)$ and a signature $sig_{sk_D}(\text{APR}_D)$.

$$\text{RREP: } S \;||\; D \;||\; \text{TTL} \;||\; n_1,\ldots,n_k \;||\; E_{pk_S}(\text{APR}_D) \;||\; E_{pk_S}(\text{APR}_D) \;||\; sig_{sk_D}(\text{APR}_D).$$

where, $E_{pk_S}(\text{APR}_D) = E_{pk_S}(\text{APR}_k) \cdot E_{pk_D}(\prod_{j=1}^{k} r_D^j)$.

Step 4: An intermediate node n_i receiving a RREP, multiplies all its own reputation values $r_i^j, j = i+1\ldots k$, for the nodes included in the *reverse* path so far. It encrypts $\prod_{j=i+1}^{k} r_i^j$ with S's public key (pk_S) and multiplies the result with the APR field.

$$\text{RREP: } S \;||\; D \;||\; \text{TTL} \;||\; n_1,\ldots,n_k \;||\; E_{pk_S}(\text{APR}_i) \;||\; E_{pk_S}(\text{APR}_D) \;||\; sig_{sk_D}(\text{APR}_D).$$

where, $E_{pk_S}(\text{APR}_i) = E_{pk_S}(\text{APR}_{i+1}) \cdot E_{pk_D}(\prod_{j=i+1}^{k} r_i^j)$.

Step 5: For any received RREP, the source decrypts the APR field using its private key sk_S and recovers APR_1. It also decrypts $E_{pk_S}(\text{APR}_D)$ and verifies $sig_{sk_D}(\text{APR}_D)$. Upon successful verification, S accepts a route only if $\text{APR}_1 \leq \text{APR}_D$.

1. To avoid network-wide flooding of RREQs, DSR adopts the ring expansion technique [14].

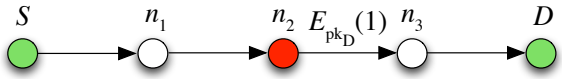

Fig. 4. Potential manipulation of the APR field. Node n_2 resets the APR field to 1 by replacing it with $E_{pk_D}(1)$ in the forward direction.

It then computes the path reputation value as

$$r_{S\to D} = \sqrt[k+1]{\frac{\text{APR}_1}{r_0}(\prod_{j=1}^{k} r_S^j)}.$$

Finally, the source selects the route with the highest $r_{S\to D}$.

We now explain all steps in detail. In Step 1, S initializes the APR field to a random value $r_0 \in (0,1]$ and encrypts it using D's public key. Therefore, only D can decrypt the APR field. Moreover, S includes a copy of the initial encrypted APR value and signs it. This copy is used by D to compute $\hat{r}_{S\to D}$ which is an intermediate path reputation value used by the destination for eliminating routes with low reputation values compared to a minimum desired reputation threshold γ_1.

Parameter r_0 is also used to detect a possible manipulation of the APR field by intermediate nodes. Such manipulation is possible because intermediate nodes are allowed to modify the APR field. For instance, consider the path shown in Figure 4. Assume that n_2 is malicious. Node n_2 may reset the APR field to 1 by replacing it with $E_{pk_D}(1)$ in the forward (backward) direction. Such a replacement will eliminate the opinions of all upstream (downstream) nodes in regards to P_{SD}. To mitigate this attack without employing expensive signature aggregation techniques, the source randomizes the APR field. Because all reputations are in $(0,1]$, it is guaranteed that $\text{APR}_k \leq r_0$. Without knowledge of r_0, the malicious node n_2 cannot properly manipulate the APR field. Random guessing of r_0 can either lead to path rejection or a route with low reputation.

In Step 2, intermediate nodes append their opinion about all nodes included in the path so far (upstream nodes). This is done by multiplying r_i^j values, encrypting them with pk_D, and multiplying the result with the encrypted APR value. Here, we employ an encryption method that implements *multiplicative homomorphism* such as RSA or Elgamal encryption [28]. An encryption method is said to be multiplicatively homomorphic if for two plaintext values a, b, it holds that, $E_{pk_i}(a) \cdot E_{pk_i}(b) = E_{pk_i}(a \cdot b)$. By performing the computations of the path reputation on the ciphertext domain, intermediate nodes receiving RREQ or RREP messages do not get to know the current path reputation estimate, or the value of r_0. Hence, the individual evaluations of intermediate nodes with respect to their peers remain secret.

In Step 3, D computes intermediate path reputations $\hat{r}_{S\to D}$ for all received RREQs. These intermediate path reputations are used to reject all paths with $\hat{r}_{S\to D} < \gamma_1$. The minimum acceptable reputation γ_1 is universally preset in the network. Because of the multiplication operation applied on the APR field, paths with $\hat{r}_{S\to D} < \gamma_1$ cannot have a final path reputation value $r_{S\to D} \geq \gamma_1$. Hence, such paths are rejected by D, in order to reduce the communication overhead of unicasting RREPs. In addition, the destination rejects all paths that exceed the length of the shortest path by a factor of λ. We call this factor as the *route selection factor* because it determines the set of paths from which the most

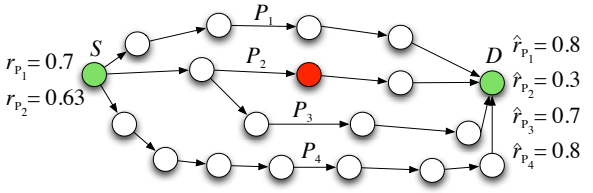

Fig. 5. Example of the DSR-based route discovery, for $\lambda = 1.2$ and $\gamma_1 = 0.6$. The destination rejects P_2 and P_4 because $\hat{r}_{P_2} < \gamma_1$ and $|P_4| > \lambda|P_2|$. Path P_1 is preferred because $r_{P_1} > r_{P_3}$.

trustworthy path is selected. Here, λ controls the tradeoff between path reputation and route length.

In Step 4, RREPs travel on multiple reverse paths to S. Intermediate nodes receiving a RREP, multiply the APR value with the reputations of all the nodes located downstream towards the destination. Similarly to the RREQ operation, this multiplication occurs on the ciphertext domain using S's public key, thus hiding the APR value from intermediate nodes.

On the final step, S decrypts the APR field using sk_S and computes $r_{S\to D}$ for all RREPs for which $\text{APR}_1 \leq \text{APR}_D$. The source then selects the path with the highest path reputation, as the single routing path to D. We emphasize that several alternative selection strategies are possible. For instance, instead of selecting the most reputable path, the source can select the shortest path with $r_{S\to D} \geq \gamma_1$. Using this strategy, shorter routes are preferred as long as they meet a minimum reputation threshold.

An example of the route discovery process outlined in Steps 1-6 is shown in Figure 5. After the RREQs have been forwarded to the destination, D considers paths $P_1 - P_4$. Path P_4 is rejected because $|P_4| > \lambda|P_2|$, where P_2 is the shortest discovered path. Moreover, path P_2 is also rejected because $\hat{r}_{P_2} < \gamma_1$ (where $\gamma_1 = 0.6$). D replies with a RREP over P_1 and P_3. At the source, P_1 is selected because $r_{P_1} > r_{P_3}$.

6 The Audit Module

The audit module is responsible for identifying the set of nodes that misbehave in a particular path P_{SD}. The source invokes the audit module if it detects poor performance on P_{SD}. The exact definition of what constitutes poor performance can be determined on the basis of a specific application running between S and D. One possible mechanism for determining the path performance is to monitor the average end-to-end packet rate $\rho_{S\to D}$ over a window of time τ. If $\rho_{S\to D} < \gamma_2$ packets per second, the audit module is activated. The threshold γ_2 is source-defined and can be statistically derived based on prior interactions of the source with other destinations, or some minimum expected network performance. The rate $\rho_{S\to D}$ can be calculated at S either by taking into account transport layer end-to-end acknowledgements, or explicit feedback provided periodically by D.

When poor performance is detected over P_{SD}, the source requests from a subset of intermediate nodes to record a digest of the set of packets they forward to the next hop. We call this, the *audit process*. Although misbehaving nodes can lie when audited, audit replies from honest nodes lead to the identification of those lies, and eventually of the misbehaving nodes. We map the audit process to Rényi-Ulam searching games [25], [29].

6.1 Rényi-Ulam Games

Rényi-Ulam games involve two players, a questioner and a responder [25], [29]. 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 (Ω, q, ℓ) and the nature of their interaction. The format of the questions can be classified into two categories; cut questions and membership questions. 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 $\omega \in A$?"

Two modes are possible for the interaction between the two players; a batch mode and an 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 queries. Else, the responder wins. The questioner is said to have a "winning strategy" if it can find ω after at most q queries, independent of how the responder lies.

6.2 Mapping to Rényi-Ulam Games

In our mapping, the role of the questioner is assumed by S and D, while the role of the responder is assumed by the nodes in P_{SD}. The search space is defined as the set of nodes in P_{SD}, i.e., $\Omega = \{n_1, \ldots, n_k\}$. The responder selects a number $\omega \in \{1, \ldots, k\}$, corresponding to the node n_ω which is misbehaving. The source's goal is to determine n_ω, i.e., to locate the misbehaving node. The questions submitted by the questioner correspond to the 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 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 6(a) and 6(b) show the mapping of the misbehavior identification problem to a Rényi-Ulam game.

6.3 Rényi-Ulam Inspired Auditing Strategies

When a node n_i is audited, it provides to the source a proof of all the packets it has received and forwarded to the next hop. Using this proof, the source constructs an audit claim a_i which is a boolean variable indicating whether a node is believed to have received and forwarded packets sent from the source. Because the proof provided by the audited node contains information for all packets sent by the source via n_i, the source can evaluate the node's behavior against any desirable selective dropping pattern. This can be achieved by classifying packets into K distinct

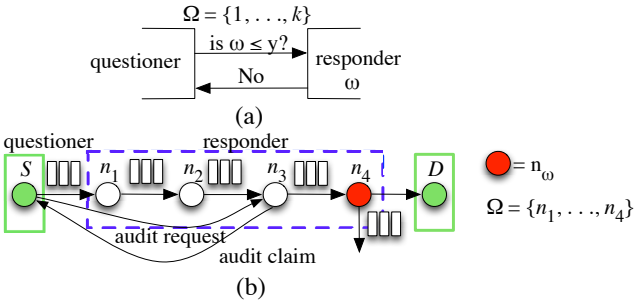

Fig. 6. (a) A generic Rényi-Ulam game. (b) Misbehavior identification mapped to a Rényi-Ulam game.

types. The importance of each packet type with respect to the application running on P_{SD} is determined by associating weights $w_i,\ i = 1 \ldots K$ with $\sum_{i=1}^{K} w_i = 1$. For an audited node n_i, the corresponding audit claim is defined as follows.

Definition 2: **Audit claim:** An audit claim a_i for a node n_i in regards to auditing a set of packets $X = \{X_1 \cup X_2 \cup \ldots \cup X_K\}$ is a claim that a set of packets $Y = \{Y_1 \cup Y_2 \cup \ldots \cup Y_K\}$ were forwarded to the next hop. Here, X_i denotes the set of packets of type i and $Y_i \subseteq X_i$. An audit claim a_i is a boolean variable

$$a_i = \begin{cases} 1, & \delta \geq \gamma_0, \\ 0, & \text{otherwise.} \end{cases}, \qquad \delta = \frac{\sum_{i=1}^{K} w_i |Y_i|}{\sum_{i=1}^{K} w_i |X_i|}, \tag{4}$$

where γ_0 denotes a desirable threshold value below which an audit claim is declared to indicate significant packet loss. If $\delta < \gamma_0$, then node n_i claims not to have forwarded packets to the next hop. An analysis on the selection of appropriate values for γ_0 and $w_i, i = 1 \ldots K$ is presented in Appendix A.

Using the audit claims of multiple audited nodes, the source attempts to first identify the misbehaving link.

Definition 3: **Misbehaving link:** a link (n_i, n_{i+1}) is said to be misbehaving if its two incident nodes n_i, n_{i+1} provide conflicting audit claims. i.e., $a_i \neq a_{i+1}$.

Proposition 1: At least one of the nodes 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, n_{i+1} forwards all packets received by n_i. Therefore, the two nodes make the same audit claims. This contradicts the definition of a misbehaving link. □

Definition 4: **Simultaneous audit:** an audit is said to be simultaneous if two or more nodes are audited with respect to the same set of packets X.

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

Proof: Since n_i and n_{i+1} are simultaneously audited and are honest, they will indicate to have received and forwarded the same set of packets. Hence, $a_i = a_{i+1}$, which contradicts the definition of a misbehaving link. Thus, n_i and n_{i+1} cannot be incident to a misbehaving link. Note that the converse statement to Corollary 1 is not true. Two nodes incident on a link providing the same audit claims cannot be assumed to be honest. □

6.3.1 Adaptive Audits with Cut Questions

We now show how the source can identify the 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_ω in P_{SD}. Define the set of nodes suspicious of misbehavior as $\mathcal{V}$. Initially, $\mathcal{V} = \{n_1, \ldots, n_k\}$. If n_i is audited and responds with $a_i = 0$, the source concludes that all nodes downstream of n_i are behaving, and therefore $\omega \leq i$. This is true since either n_i is honest in which case it never received packets in X, or $n_i = n_\omega$. Hence, $\mathcal{V}$ is reduced to $\{n_1, \ldots, n_i\}$. If n_i responds with $a_i = 1$, the source concludes that all nodes upstream of n_i are behaving, and therefore $\omega \geq i$. This is true, since if any node upstream of n_i was the misbehaving one, n_i would not have received packets in X, or $n_i = n_\omega$. Thus, $\mathcal{V} = \{n_i, \ldots, n_k\}$. We emphasize that the audited node always remains in $\mathcal{V}$, since it can lie about the forwarded packets.

Pelc [24] proposed an adaptive cut question strategy 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 $\lceil \log_2 |\Omega| \rceil + \ell + 1$ 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 $\ell + 1$ times if $\omega \leq \omega'$. Since the responder is limited in lies, the questioner can determine if $\omega' = \omega$ or a lie occurred.

We modify the winning strategy proposed by Pelc in order to identify the misbehaving link and take into account the reputation values provided by the reputation module. The source initializes $\mathcal{V} = \{n_1, \ldots, n_k\}$. Instead of performing a binary search, the source biases the search process according to the reputation values $r_S^i,\ i = 1 \ldots k$. The assumption here is that nodes with low reputation are more likely to be malicious. Therefore, we can reduce the identification delay and the communication overhead by auditing those nodes that are more likely to misbehave. The source identifies the node $n_h \in P_{SD}$ with the lowest reputation (i.e., $h = \arg\min_{\mathcal{V}} r_S^i$). It then audits node n_{h-1}. If $a_{h-1} = 1$, the source reduces $\mathcal{V} = \{n_{h-1}, \ldots n_k\}$ and immediately audits n_{h+1}. If $a_{h+1} = 0$, the source reduces $\mathcal{V} = \{n_{h-1}, n_h, n_{h+1}\}$ and audits n_h. This step concludes to the identification of the misbehaving link, either (n_{h-1}, n_h) or (n_h, n_{h+1}). Else, if either n_{h-1} or n_{h+1} provide an audit claim that excludes n_h from $\mathcal{V}$, the source updates $\mathcal{V}$ according to the answer to the cut question and repeats the identification process by selecting the (new) node in $\mathcal{V}$ with the lowest reputation value. The reputation-based search audit algorithm which we call CUT is shown in Algorithm 1.

The link identified via the reputation-based audit algorithm is guaranteed to be the misbehaving link when only one misbehaving node exists in P_{SD}. However, if $|M| \geq 2$, the source may converge on a link in which both nodes are behaving. This is possible if two (or more) nodes are colluding, as shown by the following example. In Figure 7(a), $M = \{n_1, n_4\}$, with node n_4 dropping packets initially. Assume that node n_3 has the smallest reputation value and therefore, the source audits the previous node (node n_2). Node n_2 will make an audit claim $a_2 = 1$ thus reducing $\mathcal{V} = \{n_2, n_3, n_4\}$, and removing n_1 from $\mathcal{V}$. The source then audits n_4, but n_4 lies by making an audit claim of $a_4 = 0$ (Figure 7(b)). At the same time, the n_1 assumes the role of the packet dropper. The source audits n_3 who responds with $a_3 = 0$, thus reducing $\mathcal{V}$ to $\{n_2, n_3\}$. Hence, link (n_2, n_3), is incorrectly

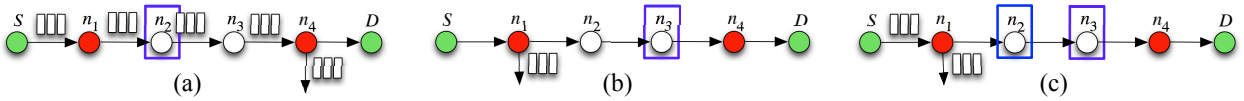

Fig. 7. (a) Nodes n_1, n_4 collude, with n_4 dropping all packets. Audited node n_2 claims misbehavior is downstream. (b) Nodes n_1, n_4 alter their behaviors, with n_1 dropping all packets. Audited node n_3 claims misbehavior is upstream. (c) Source simultaneously audits n_2, n_3 to verify if misbehaving link exists.

Algorithm 1 Reputation-based Audit Algorithm (CUT)

Initialize: $\mathcal{V} = \{n_l, \ldots, n_r\}$, $n_l \leftarrow n_1, n_r \leftarrow n_k$
while $|\mathcal{V}| > 2$
{ $h \leftarrow \arg\min_{\mathcal{V}} r_S^i$
  audit(n_{h-1})
  if $a_{h-1} = 0$
    $n_r \leftarrow n_{h-1}$
  else
  { audit(n_{h+1})
    if $a_{h+1} = 1$
      $n_l \leftarrow n_{h+1}$
    else
      { audit(n_h)
        if $a_h = 0$
          $n_r \leftarrow n_h$
        else
          $n_l \leftarrow n_h$ }}}
audit(n_l, n_r)
if $a_l \neq a_r$
  return n_l, n_r
else
  return $|M| \geq 2, Partition\ P_{SD}$

identified as the misbehaving one.

To solve this problem, we perform a simultaneous audit on the nodes of an identified misbehaving link $\mathcal{V} = \{n_i, n_{i+1}\}$. If $a_i \neq a_{i+1}$, a misbehaving link is identified. Otherwise, the source concludes that more than one misbehaving nodes exists in P_{SD}. Returning to our previous example, in Figure 7(c), n_2, n_3 are simultaneously audited. Since both nodes are honest, they make identical audit claims ($a_2 = a_3 = 0$) 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 identifies a lie occurred since n_2 has switched its audit claim from one to zero.

When the source identifies a lie occurred, it can also reach to the following conclusion: either (a) $n_\omega = n_2$ 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 path performance will be restored. In such a case, the source will take two steps. First, any outstanding audits are discarded. Second, the search is 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.

In the case of $|M| \geq 2$, let the link identified by the audit process be (n_i, n_{i+1}). Since the source knows that at least one misbehaving node is upstream of n_{i+1} and one misbehaving node is downstream of n_i, 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 $P_{n_{i+1}n_k} = \{n_{i+1}, \ldots, n_k\}$. The source repeats the auditing strategy recursively for each path partition $P_{Sn_i}, P_{n_{i+1}n_k}$. However, note that the source can only determine if misbehavior occurs in P_{SD}; not which partition.

To treat each partition separately, 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_{i+1},n_k}. Note that if n_i is the misbehaving node, it has only two strategies, (a) respond honestly, or (b) lie. If n_i lies by indicating that it has forwarded all packets to the next hop while being the packet dropper, the search on P_{Sn_i} will converge on link (n_{i-1}, n_i) with both nodes claiming to be benign.

Considering partition $P_{n_{i+1}n_k}$, if n_{i+1} also lies by claiming to have forwarded all packets to the next hop, link (n_{i+1}, n_{i+2}) will be identified as misbehaving, since n_{i+2} is honest and claims to have not received any packets. Hence, n_{i+1} is implicated to a misbehaving link and is removed from the route. Node n_i is subsequently removed, following the removal process when a path contains only a single misbehaving node, after n_{i+1} removal.

If, on the other hand, node n_i replies honestly by claiming not to have forwarded any packets, the audit process on path P_{Sn_i} will converge on misbehaving link (n_{i-1}, n_i) since n_{i-1} is honest and claims to have forwarded all packets to n_i. Similar arguments can be shown if n_{i+1} is the packet dropper or if n_i, n_{i+1} employ a mixed dropping strategy. For the case of $|M| > 2$, a similar iterative process can be employed as long as each misbehaving node is isolated on a single audited subpath.

Proposition 2: When $|M| = 1$, S converges to the misbehaving link in at most $\lceil \log_2(k) \rceil + 1$ audits when all node reputations are equal and $(k-1)$ audits when all reputations are unequal.

Proof: The proof is provided in the Appendix B. □

We emphasize that the number of misbehaving nodes $|M|$ on a single path is not known a priori. Hence, the source does not know in advance how many iterations of the auditing process will be necessary to identify the misbehaving nodes. As a result, the iterative application of the CUT strategy may yield a higher communication overhead compared to a batch strategy where all nodes of a given path are audited at once.

For most realistic scenarios, it is anticipated that few misbehaving nodes will exist in the network and that routes are likely to contain a single misbehaving node, for which case an adaptive strategy provides improved communication efficiency. An alternative strategy would be to initially perform the CUT strategy in order to identify the misbehaving link in a logarithmic number of steps. If the CUT strategy fails to converge on a single misbehaving link, thus indicating the existence of multiple misbehaving nodes, the batch strategy can be employed for further identification of misbehaving links.

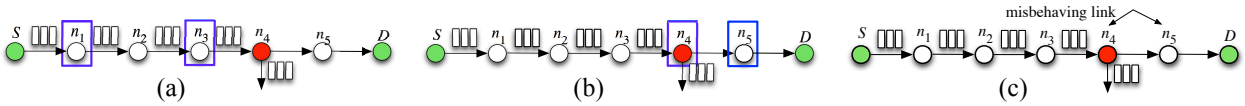

Fig. 8. (a) Let $\mathcal{V}_1 = \{n_1, \ldots, n_5\}$ with $A = \{n_1, n_2, n_3\}, B = \{n_4, n_5\}$ and $n_\omega = n_4$. The source audits A, concluding $n_\omega \notin A$. (b) The source then audits B, concluding $n_\omega \in B$. (c) The source proceeds to stage 2 with $\mathcal{V}_2 = \{n_4, n_5\}$. Since $|\mathcal{V}_2| = 2$, link $\{n_4, n_5\}$ is identified to be the misbehaving one.

6.3.2 Adaptive Audits with Membership Questions

We now describe an adaptive auditing strategy based on membership questions. Such questions can be constructed by combining two cut questions, via simultaneous audit of two nodes n_i, n_j. If nodes provide audit claims such that $a_i = a_j$, then n_i, n_j claim that $n_\omega \notin A$, since they claim to have seen similar traffic. Else, they claim $n_\omega \in A$. Compared to the CUT strategy, the membership strategy converges faster to the misbehaving link(s), at the expense of performing more audits.

Dhagat et al. [8] 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 Φ, or discards it if the answer contradicts prior answers. Suppose that $\mathcal{V}_j$ is the possible set containing ω at stage j, with $|\mathcal{V}_j| \geq 2$, and let set $\Phi = \{\phi_{j-1,a}, \phi_{j-1,b}\}$ represent the answers to stage $(j-1)$. The questioner divides $\mathcal{V}_j$ to two equal-sized subsets, A and B. The responder is asked "Is $\omega \in A$?" If the answer $\phi_{j,a}$ is "yes", the questioner adds $\{\phi_{j,a}\}$ to Φ and moves to the next stage with $\mathcal{V}_{j+1} = A$. Else, the questioner asks "Is $\omega \in B$?" If the answer $\phi_{j,b}$ is "yes," $\{\phi_{j,a}, \phi_{j,b}\}$ are added to Φ and the questioner moves to $\mathcal{V}_{j+1} = B$. If both $\phi_{j,a}, \phi_{j,b}$ are negative, the questioner removes $\{\phi_{j-1,a}, r_{\phi-1,b}\}$ from Φ and returns to stage $(j-1)$. The questioner then selects a different partition of $\mathcal{V}_{j-1}$ for stage j and repeats the questioning on each partition.

Mapping Dhagat's questioning strategy to misbehavior identification, the source begins from stage $\mathcal{V}_1 = \{n_1, \ldots, n_k\}$. Set $\mathcal{V}_1$ is divided into two subsets, $A = \{n_1, \ldots, n_i\}$ and $B = \{n_{i+1}, \ldots, n_k\}$ with $i = \lceil \frac{|\mathcal{V}_1|}{2} \rceil$. The source first asks "Is $n_\omega \in A$?" by simultaneously auditing nodes n_1, n_i. If n_1 and n_i return conflicting audit claims, i.e., $a_1 = 1, a_i = 0$, the source knows that $n_\omega \in A$, adds $\phi_{1,a}$ to Φ, and proceeds to stage two with $\mathcal{V}_2 = \{n_1, \ldots, n_i\}$. If $a_1 = a_i = 1$, the source questions "Is $n_\omega \in B$?" by simultaneously auditing nodes n_i, n_k. If n_i, n_k return $a_i \neq a_k$, the source concludes that $n_\omega \in B$, adds $\{\phi_{1,a}, \phi_{1,b}\}$ to Φ, and proceeds with $\mathcal{V}_2 = \{n_i, \ldots, n_k\}$. If $a_i = a_k$, the source concludes a lie has occurred, returns to the previous stage, and chooses another partition for $\mathcal{V}_1$.

In Figure 8(a), node n_4 is the misbehaving node. The source splits $\mathcal{V}_1 = \{n_1, \ldots, n_5\}$ to sets $A = \{n_1, n_2, n_3\}, B = \{n_4, n_5\}$, and simultaneously audits n_1, n_3 to realize the membership question "Is $n_\omega \in A$?" Since $a_1 = a_3 = 1$, the source asks "Is $n_\omega \in B$?" by simultaneously auditing n_4, n_5, as shown in Figure 8(b). The outcome of the new audit is $a_4 = 1, a_5 = 0$, and the the source concludes $n_\omega \in B$. Because $|\mathcal{V}_2| = 2$. the source converges to misbehaving link (n_4, n_5).

Proposition 3: When $|M| = 1$, the source converges to the misbehaving link in less than $4\log_2(k) + 2$ audits.

Proof: The proof is provided in the Appendix C. □

The source's auditing strategy, which we call MEM, is presented in Algorithm 2. Now assume $|M| \geq 2$. Corollary 2 states that if the source converges to a link, it must be misbehaving.

Algorithm 2 Membership Questioning Algorithm (MEM)

```
V_1 = {n_l, ..., n_r}, n_l ← S, n_r ← D, Φ = ∅
while |V_j| > 2
   {  h = ⌈|V_j|/2⌉,  φ_{1,a} = audit(n_i, n_h)
     if a_i ≠ a_j|
        Φ ← {φ_{j,a}}
        j = j + 1,  V_j = {n_l, ..., n_h}
     else
     {  φ_{j,b} = audit(n_h, n_k)
        if a_h ≠ a_r
           Φ ← {φ_{j,a}, φ_{j,b}}
           j = j + 1,  V_j = {n_h, ..., n_r}
        else
           return j = j − 1, new_partition(V_{j−1})}}}
return n_l, n_r
```

Corollary 2: The source can never converge on a link consisting of two honest nodes.

Proof: The proof is provided in the Appendix D. □

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

6.4 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. To isolate the misbehaving node, we use the ideas of *path division* and *path expansion* We first illustrate the ideas for $|M| = 1$ and then generalize to $|M| \geq 2$.

Single misbehaving node: Without loss of generality assume that the auditing process converged to a misbehaving link $(n_\omega - 1, n_\omega)$, where n_ω is the misbehaving node. The source divides P_{SD}, into two paths such that packets are routed through either n_ω or $n_{\omega+1}$, but not both, and attempts to re-identify the misbehaving link. Instead of performing the entire audit process in each of the paths, the source concentrates on the nodes around $n_{\omega-1}, n_\omega$, For example, in Figure 9(a), the source has identified link (n_3, n_4) as the misbehaving one. In Figure 9(b), the source splits the traffic between two paths that bypass n_3, n_4 in turn. Path segment $\{n_2, n_3, n_4, n_5\}$ is replaced by segments $\{n_2, n_3, n_6\}$ and $\{n_2, n_5, n_4\}$, thus isolating the behavior of n_3 and n_4. The source simultaneously audits nodes n_3, n_6 and n_5, n_4 to identify the misbehaving link. The source identifies link (n_5, n_4)

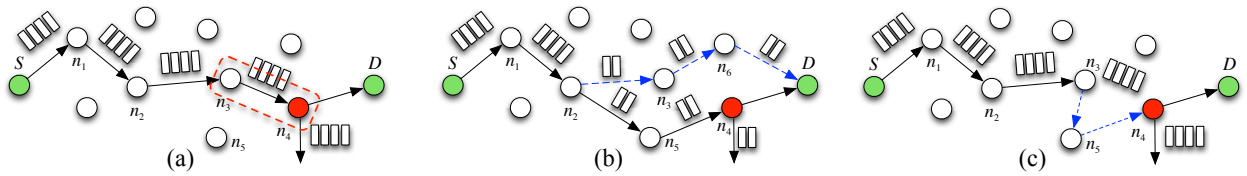

Fig. 9. (a) Node n_4 drops packets, with link (n_3, n_4) being identified as the misbehaving link, (b) P_{SD} is split to two paths, each containing one of n_3, n_4, (c) node n_5 is added to P_{SD} in order to split link (n_3, n_4).

as misbehaving, and hence identifies node n_4 as the misbehaving node, assuming that the newly added nodes are not misbehaving.

If path division cannot be performed, we employ path expansion to remove the misbehaving link (n_i, n_{i+1}) from P_{SD}. Path expansion inserts at least one node between the suspicious nodes n_i and n_{i+1}, thus forcing misbehavior to appear on a link other than (n_i, n_{i+1}). Revisiting the scenario of Figure 9(a), the source performs path expansion by adding node n_5, between n_3 and n_4, as shown in Figure 9(c). If the audit of n_5 returns an audit claim of $a_5 = 1$, node n_4 is identified as the misbehaving node. Otherwise, n_3 is assumed to be the packet dropper.

Multiple misbehaving nodes: Let us now assume the existence of multiple misbehaving nodes in P_{SD}, i.e., $|M| \geq 2$. 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 path division or path expansion in each subpath as in the case of a single misbehaving node and identify where misbehavior occurs. Note that, as in the case of $|M| = 1$, 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 $\mathcal{V}_j$ of neighboring misbehaving nodes with set $\mathcal{V}_j$ containing at most one honest node. To identify the misbehaving nodes, all nodes in $\mathcal{V}_j$ must be excluded in turn from P_{SD} according to the path division/expansion process. If $\mathcal{V}$ cannot be further reduced, the audit module reports all nodes in $\mathcal{V}$ as misbehaving and their reputation is updated accordingly by the reputation module.

6.5 Mobility

Let us now consider the case where P_{SD} changes during the audit process due to mobility. Consider first a node n_i being removed from P_{SD}. If $n_i \notin \mathcal{V}$, its removal has no effect on the auditing strategy (n_i was not suspicious of misbehavior). If $n_i \in \mathcal{V}$, either n_i is a well-behaving node in which case removing it is analogous to reducing $\mathcal{V}$, or n_i is the misbehaving node, in which case the performance in P_{SD} is restored.

Let us now look at the case where a node n_i is added 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, then it should have been excluded from $\mathcal{V}$ anyway. If n_i is a misbehaving node, then this is equivalent to the situation in which $|M| \geq 2$ and one of the misbehaving nodes has been removed from $\mathcal{V}$. However, we have shown that both auditing strategies can recognize this scenario.

6.6 The Audit Mechanism

The goal of auditing a node n_i is to force n_i to commit to the set of packets Y that it received and forwarded to the next hop. To respond to an audit, n_i records the packets forwarded for a period of time using Bloom filters which provide a storage-efficient way of performing membership testing [4]. Because a node itself is responsible for recording the packets it relays, no additional overhead is incurred during an audit. The communication overhead consists of a single message containing a Bloom filter, that must propagate from the audited node to the source. Using Bloom filters, the source achieves per-packet behavior evaluation without incurring per-packet overhead. The audit process occurs in three steps: (a) sending an audit request, (b) constructing the audit reply, and (c) computing the audit claim.

Sending an audit request: To audit n_i, the source selects an audit duration t_a measured in packets, and an initial packet sequence number s_a from which the audit will begin. The value of t_a must be sufficiently long to differentiate misbehavior from normal packet loss. For each audit, the source perturbs the values t_a and s_a, thereby preventing the conjecturing of the start and duration of audits by other nodes. The audit request is routed to n_i via P_{SD}. If it is dropped by a misbehaving node, the source interprets the lack of response as $Y = \emptyset$ and hence, $a_i = 0$. This is true since either n_i is misbehaving or a misbehaving node is upstream of n_i.

Constructing an audit reply: When n_i is audited, it constructs a Bloom filter for the set of packets Y that it receives and forwards, starting from any received sequence number $s \geq s_a$ (in case packet s_a is dropped) until packet $s_a + t_a$. By using a Bloom filter, packets in Y are compactly represented in an m-bit vector v_i with $m \ll |Y|$ [4]. For an empty set Y, all m bits of v_i are initialized to zero. To add a member $y \in Y$ to the Bloom filter, y is passed through z independent hash functions $h_j, 1 \leq j \leq z$ with each h_j having a range of $\{1, \ldots, m\}$. The corresponding bits $h_j(y)$ of vector v_i are set to one. To check if $x \in X$ is a member of Y, element x is hashed z times using the hash functions h_j and the corresponding bits are checked against the vector v_i. If a zero is found at a corresponding location in v_i, then $x \notin Y$. Else $x \in Y$ with a very high probability. After entering all packets received between times s_a and $s_a + t_a$ in 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 Y that it claims to have forwarded to the next hop.

Computing the audit claim: To assess the behavior of audited nodes, the source verifies the authenticity of v_i and discards it if the signature check fails. Otherwise, it computes the value of δ in (4), which requires the evaluation of $|Y_j|$, for each packet type $j = 1 \ldots K$. While a direct membership testing on v_i is possible, this would require the storage of packets at the source. To eliminate this storage overhead, the source creates its own Bloom filters $v_{S_j}, j = 1 \ldots K$, for each packet type. It then computes $|Y_j|, \forall j$ given the Bloom filter length m, the cardinalities of sets Y, X_j,

the number of hashes z, and the inner product $\langle v_i, v_{S_j} \rangle$ [4]. Here, $|Y| = |X|$.

$$|Y_j| = |X_j \cap Y| \approx |X_j| + |Y| - \frac{\log\left(\frac{<v_i, v_{S_j}>}{m}\right) + \left(1 - \frac{1}{m}\right)^{z|X_j|} + \left(1 - \frac{1}{m}\right)^{z|Y|}\Big)}{z \log\left(1 - \frac{1}{m}\right)}. \quad (5)$$

The source then evaluates the audit claim a_i based on (4). This claim is then used by the audit module to locate the misbehaving node and update the reputation module with the evaluation of all the nodes in P_{SD}. Further, these evaluations propagate to all nodes in P_{SD} as second-hand information.

Eq. (5) enables the source to do testing against any required selective dropping patterns using the same Bloom filter provided by the audited node. This flexibility can be used to differentiate malicious packet dropping from packet dropping due to congestion or poor channel conditions, assuming that the misbehaving nodes does not mimic the channel behavior. For a well-engineered network, mimicking of dropping patterns similar to those of congestion or channel packet loss has limited effect on the network performance. To the best of our knowledge, only one prior work has attempted to address the problem of differentiating malicious from natural packet loss, where a similar assumption for the malicious node's behavior was made [26].

7 Performance Evaluation

Simulation Setup–We randomly deployed 100 nodes within an area of 100m×100m. A fraction of these nodes was randomly selected to misbehave. The misbehaving nodes independently implemented a packet dropping strategy, either continuous or selective. The reputation of each node was initialized to 0.5. We randomly selected 1,000 source/destination pairs from the set of honest nodes. For each pair, we ran the route discovery module to construct a trustworthy path. In each session, the source routed 10,000 packets to the destination via the established path. To isolate the performance degradation due to malicious dropping, lower layer details such as contention and retransmissions due to collisions were abstracted. In our simulations, packet dropping due to channel conditions was implemented as a dropping module on each node, similar to that of misbehavior. Each experiment was repeated for 50 random network topologies. All our simulations were performed using a stand-alone *Java*-based simulator that implemented the AMD module at every node of the network [32].

Performance Evaluation Metrics–The following metrics were used to evaluate the performance of AMD.

Average reputation value $r^i(t)$: Average reputation of node n_i at time epoch t, as evaluated by all other nodes.

Percentage of dropped packets Fr: Percentage of packets dropped by any intermediate node (network-wide) over the number of packets sent by any source in an interval of 50 epochs.

Average route expansion factor E: Length of the path P_{SD} discovered by AMD, over the length of the shortest path, averaged over all paths discovered during the course of the simulation.

Normalized route discovery overhead: Communication overhead of AMD route discovery normalized over the communication overhead of DSR, averaged over all paths discovered during the course of the simulation.

Route discovery latency overhead: Latency overhead of the route discovery process due to cryptographic operations, measured in seconds.

Audit communication overhead: Number of messages transmitted/received for identifying the misbehaving node(s).

Identification delay: Time elapsed from the occurrence of misbehavior until the misbehaving nodes are identified, measured in number of audits.

7.1 Evolution of Reputation Values

In the first set of experiments, we evaluated the evolution of the average reputation values as a function of time and for different parameters α, β (see eq. (1)), when 20% of the nodes are continuously misbehaving. Figure 10(a) shows the average reputation value of honest and misbehaving nodes for different α, β, as a function of the simulation time in epochs. We observe that the reputation value of malicious nodes rapidly decreases with the progress of time, even for large values of α. On the other hand, the reputation value of honest nodes progressively approaches the maximum value of one.

Figure 10(b) shows the average reputation value of well-behaving nodes experiencing 30% packet loss due to poor channel conditions for different thresholds γ_0. We observe that the reputation value increases slowly when $\gamma_0 = 0.6$, while it follows a slow downward trend for larger γ_0's. While nodes experiencing poor channel conditions are not misbehaving, such nodes are dropping too many packets and should not be included in routing paths. Hence, their reputation should be lowered.

Figure 10(c) shows the average reputation value of misbehaving nodes, when such nodes randomly drop a percentage of their incoming traffic. We observe that the reputation of such nodes approaches zero, though the decrease is slower compared to the continuous misbehavior case. In Figure 10(d), we show the average reputation value of misbehaving nodes when they employ a selective dropping strategy. In this experiment, we considered packets of type I (e.g., control packets), and type II (e.g., data packets). The traffic stream consisted of 30% type I packets and 70% type II packets. The source weighted each packet type with varying weights w_1, w_2. The misbehaving nodes were assumed to drop packets of type I with probability 0.9 and packets of type II with probability 0.1. From Figure 10(d), it is evident that when the weight distribution is $w_1 = 0.9, w_2 = 0.1$, the reputation of the packet droppers rapidly approaches zero. On the other hand, for a weight distribution of $\{0.6, 0.4\}$, the reputation value of droppers exhibits an oscillating behavior with a downward trend. While not being equal to zero, the reputation value of of misbehaving nodes is still comparatively lower to that of well-behaving ones, thus leading to the exclusion of the former from routing paths.

7.2 Evaluation of the Percentage of Dropped Packets

In this set of experiments, we compared the performance of AMD with the performance of DSR in the percentage of dropped

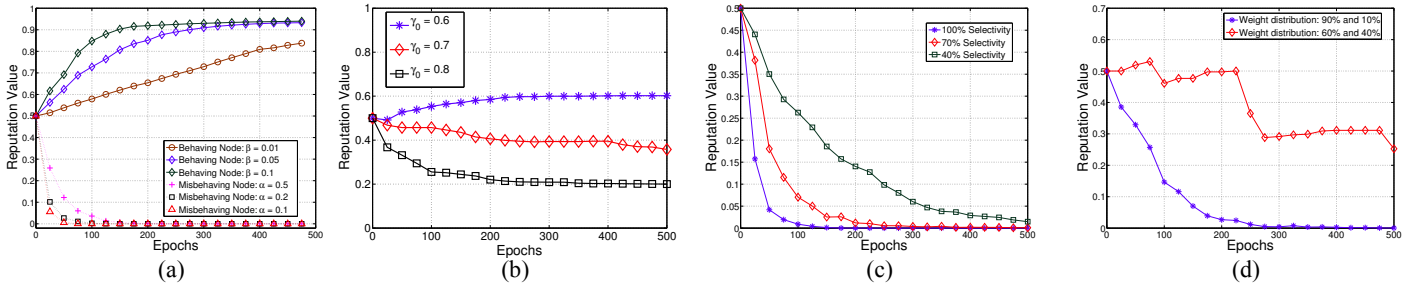

Fig. 10. (a) Average reputation of well-behaving and misbehaving nodes for different α, β, as a function of time for continuous misbehavior, (b) average reputation of well-behaving nodes experiencing packet loss with probability 0.3, (c) average reputation value of misbehaving nodes under a random dropping strategy, (d) average reputation of misbehaving nodes for different weight distributions.

packets due to misbehavior. Figure 11(a) shows Fr as a function of time, when misbehaving nodes continuously drop packets, for different percentages of misbehaving nodes (10%, 20%, 30%). As expected, the percentage of dropped packets for DSR is fairly constant, since DSR is not designed to avoid nodes with poor reputation. The small observed variation is related to the random selection of source/destination pairs. On the other hand, AMD quickly reduces the percentage of dropped packets to almost zero. This is due to the fact that misbehaving nodes are excluded from the routing paths as their reputation value decreases. Hence, most routes consist only of honest nodes. We further implemented a random packet dropping behavior in which misbehaving nodes randomly dropped a fraction of the traffic. Figure 11(b) shows Fr as a function of time when 20% of the nodes are misbehaving. We observe that a less aggressive behavior can only delay the isolation of misbehaving nodes.

7.3 Evaluation of the AMD Overhead

Average Route Expansion Factor: Figure 11(c) shows the average route expansion factor E as a function of the percentage of misbehaving nodes, for various values of the route selection factor λ. The value of λ represents the extend to which AMD expands the search for the most reputable path, relatively to the shortest path. From Figure 11(c), we observe that for $\lambda = 1.2$, the average length of paths discovered by AMD are only 4% longer than the shortest paths even when 45% of the nodes misbehave. This value increases to only 11% when the search is expanded to paths up to 60% longer than the shortest path. This performance indicates that AMD pays a relatively small penalty for avoiding the misbehaving nodes. Moreover, even when the search is expanded over long paths, shorter paths are still preferred due to the multiplicative nature of the reputation path value.

Figure 11(d) shows the average path reputation value of the paths discovered by AMD as a function of the route selection factor λ and for different number of misbehaving nodes. We observe that increase of λ marginally increases the reputation of the paths discovered by AMD. This reinforces the fact that a small value of λ should be sufficient to discover reputable paths.

Communication Overhead of Route Discovery: Figure 12(a) shows the communication overhead of AMD normalized over the overhead of DSR, as a function of the nodes' communication range r, for various values of the route selection factor λ. We observe that the route discovery phase of AMD is as much as 2.5 times more expensive than that of DSR for large λ. This is due to the additional flooded messages of the RREQ phase, and the multipath nature of the RREP phase. However, as it is indicated by Figures 11(c) and 11(d), a value of $\lambda = 1.2$ is sufficient to discover reputable paths (the average length of the paths discovered by AMD was within 10% of the shortest path). For $\lambda = 1.2$, the normalized communication overhead varies from 1.7 when $r = 12$ to 1.05 when $r = 22$, indicating a fairly efficient route discovery of trustworthy paths.

TABLE 1
Typical timings for performing various public key operations in mobile devices [12], [30].

parameter	delay (ms)	parameter	delay (ms)
t_{enc} (RSA, 1024-bit)	29.0 [12]	t_{dec} (RSA, 1024-bit)	151.0 [12]
t_{sig} (ECC, 163-bit)	5.7 [30]	t_{ver} (ECC, 163-bit)	17.9 [30]

Route discovery latency overhead: We further evaluated the latency of AMD due to the cryptographic operations executed during the route discovery phase. Let $t_{enc}, t_{dec}, t_{sig}$, and t_{ver} denote the required time for performing a public key encryption, a public key decryption, a signature generation, and a signature verification, respectively. Consider the process of discovering a path with $(k-1)$ intermediate nodes. Referring to the steps of the in Section 5.1, the delay overhead is as follows.

In Step 1, the source performs one encryption and generates one signature. In Step 2, every intermediate node performs one encryption in order to compute the accumulated path reputation (APR) field. In Step 3, the destination performs one decryption, one signature verification, one encryption, and one signature generation. In Step 4, every intermediate node performs one encryption. In Step 5, the source performs one decryption and one signature verification. The total delay for steps 1–5 is $T_{RD} = 2(t_{sig} + t_{ver} + kt_{enc} + t_{dec})$. Typical times required to perform various public key operations during the route discovery phase are shown in Table 1 [12], [30].

We note that while encryption and decryption of the APR field must be performed with a cryptosystem that satisfies the homomorphic multiplication property, signing r_0 need not present any homomorphism. Therefore, the latter can be implemented using efficient public key cryptosystems such as elliptic curves. Using the values shown in Table 1, a route of 10 hops would incur a delay overhead of 929.2 ms. We emphasize that AMD does not impose any latency overhead for data forwarding. AMD can operate if end-to-end data transfers are encrypted or unencrypted.

Communication Overhead of the Audit Module: We compared AMD with the CONFIDANT scheme [5] from the class of reputation-based systems and the 2ACK [19], and ODSBR

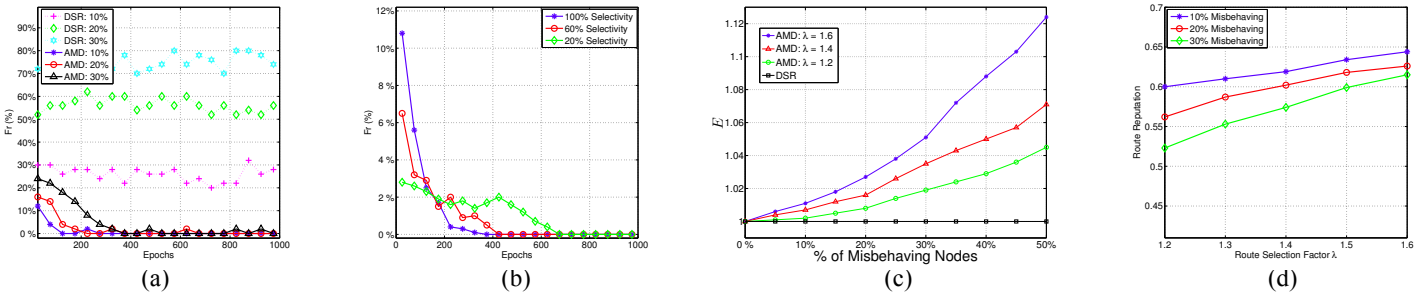

Fig. 11. (a) Fr as a function of time under continuous misbehavior, (b) Fr as a function of time under a random dropping strategy, (c) the average route expansion factor E as a function of the percentage of misbehaving nodes, for various λ, (d) the average path reputation value of paths discovered by AMD as a function of λ for different percentage of misbehaving nodes

[2] schemes from the class of acknowledgment-based schemes. We evaluated the communication overhead and delay associated with the identification of compromised nodes. In the CONFIDANT scheme, every one-hop neighbor of a transmitting node was assumed to operate in promiscuous mode, thus overhearing transmitted messages. The energy for overhearing a message was set to 0.5 times the energy required to transmit [10]. For 2ACK, a fraction p of the messages transmitted by each node was acknowledged two hops upstream of the receiving node. We set that fraction to $p = \{1, 0.5, 0.1\}$. In ODSBR, audited nodes acknowledged audited packets back to the source. CONFIDANT, 2ACK, and ODSBR are proactive, incurring communication overhead regardless of the existence of misbehavior. However, AMD incurs overhead only if misbehavior exists, due to its reactive nature. The audit duration for AMD was set to t_a=200 packets.

In Figure 12(b), we show the communication overhead in messages, as a function of the percentage of misbehaving nodes. The Y axis is in logarithmic scale. We observe that communication overhead of AMD is significantly lower compared to other schemes due to the per-flow nature of the behavior evaluation. In Figure 12(c), we show the communication overhead as a function of audit size, t_a. All three schemes incur a linear increase in communication overhead with the audit size. On the other hand, the overhead of AMD depends on the number of audits, not the duration of each audit. Hence, it is independent of the audit size.

Identification Delay: In Figure 12(d), we show the identification delay as a function of the path length for the CUT and MEM algorithms, measured in the number of audits until the misbehaving link is identified. We observe that both CUT and MEM incur approximately the same delay due to their binary search nature. In Figure 12(e), we show the delay required to identify two misbehaving nodes as a function of the path length. In the CUT algorithm, after the path is partitioned, the auditing of the two partitions is dependent on the strategies of the misbehaving nodes. If only one misbehaving node drops packets at a time, CUT 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 partition contains a pseudo-source and a pseudo-destination, thus decreasing the identification delay.

For CUT, we plot both the case of search in series and in parallel. 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 fact that 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.

In Figure 12(f), we compare the identification delay of AMD with other schemes. To provide a fair comparison, we have assumed that a node is assessed by evaluating its behavior over a fixed number of packets (audit duration) for all schemes. Hence, a 2ACK with $p = 0.1$ would have to monitor a node ten times longer compared to the case of $p = 1$, in order to evaluate its behavior. We observe the logarithmic increase of identification delay with the path length for the AMD and ODSBR schemes. CONFIDANT requires a single audit duration to identify misbehavior. 2ACK also requires a single audit duration when all packets are acknowledged, while the delay increases multi-fold when a fraction of packets is acknowledged upstream.

8 Conclusions

We developed AMD, a comprehensive misbehavior detection and mitigation system which integrates three critical functions: reputation management, route discovery, and identification of misbehaving nodes via behavioral audits. We modeled the process of identifying misbehaving nodes as Rényi-Ulam games and derived resource-efficient identification strategies. We showed that AMD recovers the network operation even if a large fraction of nodes is misbehaving at a significantly lower communication cost. Moreover AMD can detect selective dropping attacks over end-to-end encrypted traffic streams.

References

[1] G. Acs, L. Buttyan, and L. Dora. Misbehaving router detection in link-state routing for wireless mesh networks. In *Proc. of WoWMoM*, pages 1–6, 2010.
[2] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens. ODSBR: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks. *ACM Transactions on Information System Security*, 10(4):11–35, 2008.
[3] K. Balakrishnan, J. Deng, and P. K. Varshney. Twoack: Preventing selfishness in mobile ad hoc networks. In *Proc. of WCNC*, 2005.
[4] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
[5] S. Buchegger and J.-Y. L. Boudec. Self-policing mobile ad-hoc networks by reputation systems. *IEEE Comm. Magazine*, pages 101–107, 2005.
[6] L. Buttyan and J.-P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *Mobile Net. and Applications*, 8(5):579–592, 2003.

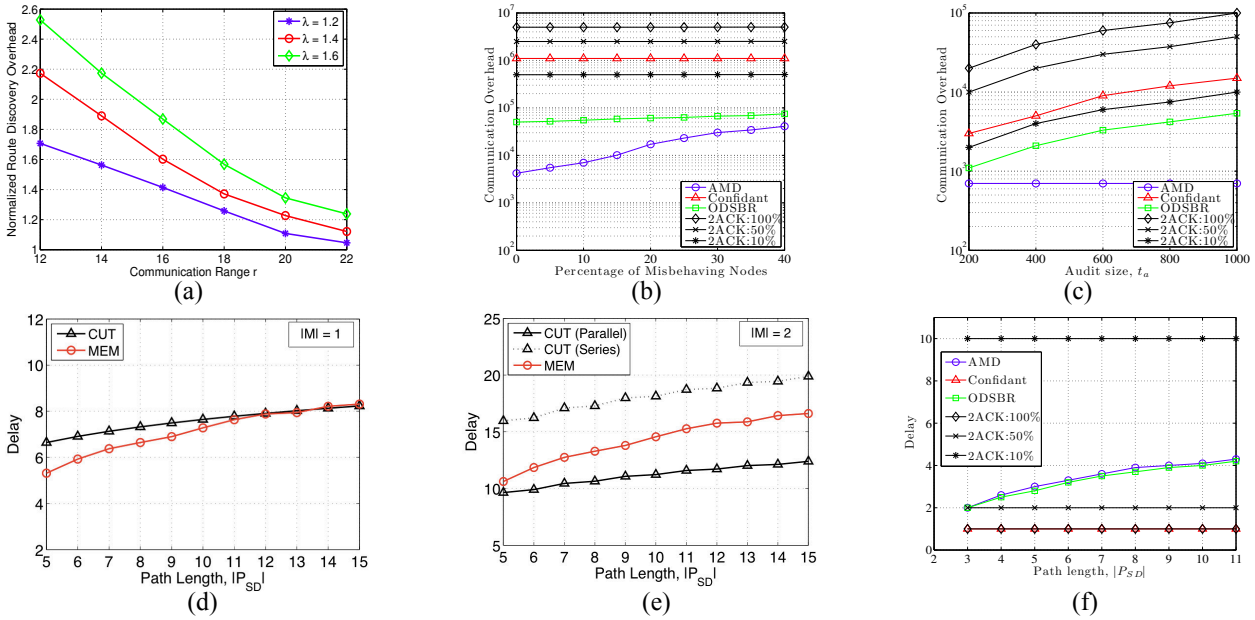

Fig. 12. (a) the communication overhead of AMD normalized over the overhead of DSR as a function of the nodes' communication range, for various λ, (b) the communication overhead of the audit process as a function of the percentage of misbehaving nodes, (c) the communication overhead as a function of audit size, t_a, (d) identification delay as a function of the path length $|P_{SD}|$, in number of audits, for (d) one misbehaving node, (e) two misbehaving nodes, (f) comparison with other schemes.

[7] J. Crowcroft, R. Gibbens, F. Kelly, and S. Östring. Modelling incentives for collaboration in mobile ad hoc networks. In *Proc. of WiOpt*, 2003.

[8] A. Dhagat, P. Gács, and P. Winkler. On playing "twenty questions" with a liar. In *Proc. of SODA*, pages 16–22, 1992.

[9] Y. Dong, H. Go, A. Sui, V. Li, L. Hui, and S. Yiu. Providing distributed certificate authority service in mobile ad hoc networks. In *Proc. of SecureComm 2005*, pages 149–156, 2005.

[10] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proc. of INFOCOM*, pages 1548–1557, 2001.

[11] S. Ganeriwal, L. Balzano, and M. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Transctions on Sensor Networks*, 4(3):1–37, 2008.

[12] K. Hansen, T. Larsen, and K. Olsen. On the efficiency of fast rsa variants in modern mobile phones. *Arxiv preprint arXiv:1001.2249*, 2010.

[13] Q. He, D. Wu, and P. Khosla. SORI: A secure and objective reputation-based incentive scheme for ad hoc networks. In *Proc. of WCNC*, 2004.

[14] D. Johnson, D. Maltz, and Y.-C. Hu. The dynamic source routing protocol for mobile ad hoc networks (DSR). *draft-ietf-manet-dsr-09.txt*, 2003.

[15] A. Jøsang and R. Ismail. The beta reputation system. In *Proc. of the 15th Bled Electronic Commerce Conference*, pages 324–337, 2002.

[16] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *Ad hoc networks*, 1(2-3):293–315, 2003.

[17] W. Kozma and L. Lazos. Dealing with liars: Misbehavior identification via Rényi-Ulam games. *Security and Privacy in Communication Networks*, pages 207–227, 2009.

[18] W. Kozma Jr. and L. Lazos. REAct: Resource-efficient accountability for node misbehavior in ad hoc networks based on random audits. In *Proc. of WiSec*, 2009.

[19] K. Liu, J. Deng, P. Varshney, and K. Balakrishnan. An acknowledgment-based approach for the detection of routing misbehavior in manets. *IEEE Transactions on Mobile Computing*, 6(5):536–550, 2007.

[20] Y. Liu and Y. R. Yang. Reputation propagation and agreement in mobile ad-hoc networks. In *Proc. of IEEE WCNC*, pages 1510–1515, 2003.

[21] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proc. of MobiCom*, pages 255–265, 2000.

[22] P. Michiardi and R. Molva. CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Proc. of CMS*, pages 107–121, 2002.

[23] V.-N. Padmanabhan and D.-R. Simon. Secure traceroute to detect faulty or malicious routing. *SIGCOMM CCR*, 33(1), 2003.

[24] A. Pelc. Detecting errors in searching games. *Journal of Combinatorial Theory Series A*, 51(1):43–54, 1989.

[25] A. Rényi. *A Diary on Information Theory*. Wiley, New York, 1984.

[26] T. Shu and M. Krunz. Detection of malicious packet dropping in wireless ad hoc networks based on privacy-preserving public auditing. In *Proc.of WiSec*, pages 87–98, 2012.

[27] S. Soltanali, S. Pirahesh, S. Niksefat, and M. Sabaei. An Efficient Scheme to Motivate Cooperation in Mobile Ad hoc Networks. In *Proc. of ICNS*, pages 92–98, 2007.

[28] D. Stinson. *Cryptography: theory and practice*. CRC press, 2006.

[29] S. Ulam. *Adventures of a Mathematician*. Scribner, New York, 1976.

[30] D. Westhoff, B. Lamparter, C. Paar, and A. Weimerskirch. On digital signatures in ad hoc networks. *European Transactions on Telecommunications*, 16(5):411–425, 2005.

[31] Y. Xue and K. Nahrstedt. Providing fault-tolerant ad-hoc routing service in adversarial environments. *Wireless Personal Communications, Special Issue on Security for Next Generation Communications*, 29(3-4):367–388, 2004.

[32] Y. Zhang and L. Lazos. http://www2.engr.arizona.edu/~llazos/misc/AMDsimulator.zip, 2012.

[33] S. Zhong, J. Chen, and Y. R. Yang. Sprite: A simple cheat-proof, credit-based system for mobile ad-hoc networks. In *Proc. of INFOCOM*, pages 1987–1997, 2003.

Yu Zhang is a research assistant at the Department of Electrical and Computer Engineering, University of Arizona, Tucson. His research interests are in the area of network security for wireless networks with emphasis on misbehavior detection.

Loukas Lazos is an Assistant Professor of Electrical and Computer Engineering at the University of Arizona. His main research interests are in the areas of networking, security, and wireless communications, focusing on the identification, modeling, and mitigation of security vulnerabilities, visualization of network threats, and analysis of network performance. He is a recipient of the Faculty Early Career Awards including NSF CAREER (2009), for his research in security of multi-channel wireless networks.

William Jr. Kozma is a research assistant at the Dept. of Electrical and Computer Engineering, University of Arizona, Tucson. His research interests are in the area of protocol development for secure communications in wireless ad hoc and sensor networks.