REAct: Resource-Efficient Accountability for Node Misbehavior in Ad Hoc Networks based on Random Audits

William Kozma Jr.
University of Arizona
Electrical and Computer Engineering Dept.
Tucson, Arizona 85721
wkozma@ece.arizona.edu

Loukas Lazos
University of Arizona
Electrical and Computer Engineering Dept.
Tucson, Arizona 85721
llazos@ece.arizona.edu

ABSTRACT

Wireless ad hoc networks rely on multi-hop routes to transport data from source to destination. The routing function is implemented in a collaborative manner, with each node responsible for relaying traffic to the destination. However, an increasingly sophisticated pool of users with easy access to commercial wireless devices, combined with the poor physical and software security of the devices, can lead to node misconfiguration or misbehavior. A misbehaving node may refuse to forward packets in order to conserve its energy (selfishness), or simply degrade network performance (maliciousness).

In this paper, we investigate the problem of uniquely identifying the set of misbehaving nodes who refuse to forward packets. We propose a novel misbehavior identification scheme called REAct that provides resource-efficient accountability for node misbehavior. REAct identifies misbehaving nodes based on a series of random audits triggered upon a performance drop. We show that a source-destination pair using REAct can identify any number of independently misbehaving nodes based on behavioral proofs provided by nodes. Proofs are constructed using Bloom filters which are storage-efficient membership structures, thus significantly reducing the communication overhead for misbehavior detection.

Categories and Subject Descriptors

C.2.0 [**Computer - Communication Networks**]: General - Security and Protection

General Terms

Security

Keywords

Misbehavior, Routing, Ad hoc Networks, Bloom Filter

WiSec'09, March 16–18, 2009, Zurich, Switzerland.

1. INTRODUCTION

Wireless ad hoc networks are characterized by the spontaneous self-organization of a collection of nodes into a multi-hop network, in the absence of a pre-deployed infrastructure. Due to their low deployment cost and self-adaptability, they find numerous civilian applications, such as collaborative computing, emergency services, vehicular networks, patient and environmental monitoring, as well as military applications, such as surveillance, and target tracking.

Ad hoc networks compensate for the lack of infrastructure via the collaborative implementation of network functionalities. Network nodes share their resources for the purpose of realizing network-wide services. For example, the routing service is implemented in a multi-hop fashion where intermediate nodes forward data from source to destination [13].

The performance of such collaborative services relies on the willingness of the network devices to participate and conform to the network protocols. However, there is no guarantee that nodes will indeed collaborate. Malicious users may modify the software or hardware of their devices in order to maximize their individual network benefit while expending minimum resources. Moreover, malicious users may attempt to degrade network performance through protocol misbehavior. At present, it is too expensive to equip every network device with tamper-proof hardware [8]. Hence, a network must be able to detect instances of misbehavior, identify the misbehaving nodes, and revoke them from the network.

In this paper, we address the problem of misbehavior in the routing protocol. Specifically, we address the problem of *identifying misbehaving nodes that refuse to forward packets to a destination.* Such misbehavior has been shown to have a severe impact on the network operability [5, 15–17].

Currently proposed solutions to the routing misbehavior problem can be classified into incentive-based schemes [7, 12, 23], reputation-based schemes, [5, 6, 11, 17, 19, 21] and acknowledgment-based schemes [1, 2, 15, 20, 22]. Reputation-based systems rely on message overhearing for the evaluation of the reputation of each node. However, operating in promiscuous mode can be almost as expensive as transmitting [9]. Furthermore, neighbor monitoring is difficult to implement in multi-channel networks where nodes may engage in parallel transmissions in orthogonal frequency bands. In acknowledgment systems, every packet is acknowledged two or more hops upstream, verifying that intermediate node(s) forwarded the packet [1, 2, 15, 20, 22]. Reputation and acknowledgment systems are proactive, incurring high energy overhead for the continuous monitoring of node behavior.

We address the problem of misbehavior with resource efficiency in mind, by considering a reactive design.

1.1 Our Contributions

We propose REAct, a resource-efficient misbehavior detection scheme, that provides publicly verifiable proof of node misbehavior. REAct is reactive in nature, thus only employed when a significant performance degradation is reported on a given routing path. We show that REAct is capable of detecting multiple independent misbehaving nodes located on the path from source to destination, through a series of random audits. Misbehaving nodes are identified based on the behavioral proofs provided by honest nodes in response to the random audits. These proofs are constructed using Bloom filters which are storage-efficient membership structures [3], thus significantly reducing the communication overhead associated with sending the proofs to the source. We show that REAct achieves significantly improved communication efficiency compared to the proactive schemes, at the expense of increased delay in misbehavior identification.

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 develop and analyze our misbehavior identification scheme. In Section 5, we compare the performance of REAct with previously proposed schemes. In Section 6, we conclude.

2. RELATED WORK

Previously proposed methods for addressing the misbehavior problem can be classified into, (a) credit-based systems [7, 12, 23], (b) reputation-based systems [5, 6, 11, 17, 19, 21], and (c) acknowledgment-based systems [1, 2, 15, 20, 22].

2.1 Credit-Based Systems

Credit systems [7, 12, 23] provide incentives for cooperation. Buttyan et al. [7] proposed a system where nodes receive credit (nuglets) for packets they forward, and spend credit to transmit their own packets. A nuglet counter records credit, while tamper proof hardware called the security module prevents the counter from becoming negative or being modified.

Zhong et al. [23] proposed Sprite, where nodes collect *receipts* for packets they forward. Nodes upload receipts are uploaded to a Credit Clearance Service (CCS) in return for credit to transmit their own packets. Jakobsson et al. [12] proposed a scheme where cryptographic payment tokens are attached to packets and managed by a base station, i.e., a form of virtual bank.

While credit-based systems motivate cooperation, malicious nodes have no incentive to collect credit and receive no punishment for non-cooperation. Furthermore, tamper proof hardware is currently too expensive to integrate in every network device [10]. Sprite removes this requirement, at the expense of a CCS. Lastly, credit systems lack a mechanism for identifying the misbehaving node(s) for revocation.

2.2 Reputation-Based Systems

Reputation systems [5, 6, 11, 17, 19, 21] use neighborhood monitoring techniques to identify misbehaving nodes. Marti et al. [17] proposed a scheme which relies on two modules, a *watchdog* and a *pathrater*. The watchdog module monitors the behavior of neighboring nodes by operating its radio in promiscuous mode to verify packet forwarding, making accusations of misbehavior when packets are not forwarded. The pathrater module uses the accusations generated to select paths that will most likely avoid misbehavior.

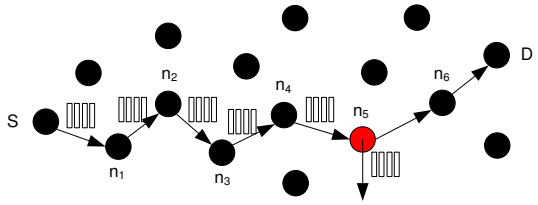

Figure 1: Node S is sending traffic to D along P_{SD}. Node n_5 drops all packets.

Buchegger et al. [5,6] proposed the *CONFIDANT* scheme, utilizing the watchdog/pathrater model, where detected misbehavior is broadcast using alarm messages. He et al. [11] proposed SORI, which propagates monitored behavior, thus relying on first- and second-hand information. Michiardi et al. [19] proposed CORE, assigning reputation metrics by combining observations, positive reports by neighbors, and task-specific behavior. Paul et al. [21] proposed using routing message verification and packet comparisons to identify misbehavior.

Neighborhood 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 or different sectors, thus unable to monitor their peers. Moreover, operating in promiscuous mode requires up to 0.5 times the energy for transmitting a message [9], thus making message overhearing an energy-expensive operation. Finally, reputation-based systems are proactive in nature, requiring the constant monitoring of nearby nodes for the building of reputation metrics. Hence, overhead is incurred on all nodes regardless of whether a misbehaving node exists.

2.3 Acknowledgment-Based Systems

Acknowledgment systems [1, 2, 15, 20, 22] rely on acknowledgments to verify that packets were forwarded. Liu et al. [15] proposed the 2ACK scheme, where nodes explicitly send 2-hop acknowledgments in the reverse direction to verify the cooperation of the intermediate node in packet forwarding. A value is assigned to the quantity/frequency of un-verified packets to determine misbehavior. Padmanabhan et al. [20] proposed a method based on traceroute in which the source probes the path with pilot packets indistinguishable from data packets. Xue et al. [22] proposed Best-effort Fault-Tolerant Routing, relying on end-to-end ACK's to monitor packet delivery ratio and select routing paths that minimize misbehavior.

Acknowledgment-based schemes are proactive, and hence incur message overhead regardless of the presence of misbehavior. 2ACK provides a method to reduce message overhead by acknowledging only a fraction of the packets, with the tradeoff of increased delay in misbehavior detection.

3. MODELS AND PROBLEM STATEMENT

Network Model: We assume a multi-hop ad hoc network consisting of N nodes. Each node is responsible for relaying messages from source S to destination D. We as-

sume S is aware of nodes in path P_{SD}, as in Dynamic Source Routing (DSR) [13]. If DSR is not used, the source can identify the nodes in P_{SD} by performing a traceroute operation. For simplicity, we number the nodes in $P_{SD} = \{n_1, \ldots, n_k\}$ in ascending order with $k = |P_{SD}|$. Node n_i is *upstream* of n_j if $i < j$ and is *downstream* of n_j if $i > j$.

We also assume the source receives feedback from the destination when a significant performance drop in metrics of interest, such as throughput or delay occurs [18]. We assume that message integrity and authenticity can be verified using resource-efficient cryptographic methods, i.e., nodes may use the Elliptic Curve Digital Signature Algorithm (ECDSA) that has been shown feasible for resource limited devices such as sensors [14]. Finally, we assume there are at least two independent paths to any destination, i.e., the network is two-connected. This assumption is essential for reaching every node in P_{SD} through a disjoint path.

Adversarial Model: We assume the existence of multiple independently misbehaving nodes in P_{SD}. Any node in P_{SD} may be misbehaving, except the source and the destination which are assumed to be trusted. The goal of misbehaving nodes is to degrade throughput while remaining undetected. Misbehaving nodes are assumed to be aware of the mechanisms used for misbehavior detection. The case of multiple colluding nodes is left as future work.

Problem Statement: Consider a path P_{SD} from source S to destination D which contains a set M of independently misbehaving nodes with $S, D \notin M$. Nodes in M misbehave by (a) dropping all packets routed through them or, (b) selectively dropping packets. *We address the problem of uniquely identifying set M such that publicly verifiable proof of the misbehavior is constructed in order to remove M from the network.* In Figure 1, S is sending packets to D through $P_{SD} = \{n_1, \ldots, n_6\}$. Node n_5 misbehaves by dropping all packets routed through it. Our goal is to identify n_5 and provide evidence of its misbehavior.

4. THE REACT SCHEME

REAct (Resource-Efficient ACcounTability) is a reactive misbehavior detection scheme triggered if a performance degradation is experienced on path P_{SD}. Destination D notifies source S of a performance drop by sending an alarm message via a path disjoint to P_{SD}. Upon receipt of an alarm, S initiates a process of random audits to identify the misbehaving node. The definition of misbehavior is a parameter within our solution.

The source requests from nodes in P_{SD} to commit to a "behavioral proof", via an audit mechanism. When node $n_i \in P_{SD}$ is audited, n_i creates a record of all packets it forwards. The goal of the audits is to combine information obtained from honest nodes to identify the misbehaving ones. While misbehaving nodes may change strategies, we show that S can always identify them, given sufficient audits.

REAct consists of three phases: (a) the audit phase, (b) the search phase, and (c) the identification phase. We now describe the three phases in detail.

4.1 Audit Phase

The goal of the audit phase is to verify that the audited node n_i forwards packets to the destination. When a node is audited, it has to provide proof of the packets it forwards. The proof is used by the source S to perform a simple membership test: Did node n_i forward packets in set X to the next hop? The audit phase occurs in three steps: (a) sending an audit request, (b) constructing a behavioral proof, and (c) processing the behavioral proof.

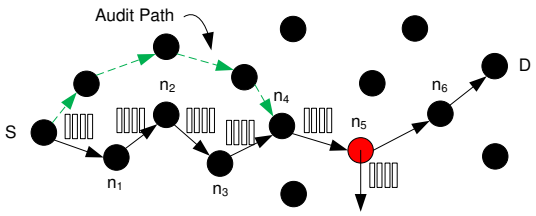

Figure 2: S selects a path P_{Sn_4} to send the audit request to n_4.

4.1.1 Sending an Audit Request

Once misbehavior has been detected in P_{SD}, the source S selects a node n_i to be audited based on the search phase (see Section 4.2). The source constructs a routing path P_{Sn_i} such that P_{Sn_i} and P_{SD} are disjoint to avoid the audit request being dropped by the misbehaving node. The source also selects an audit packet count, a_{count}, denoting the duration of the audit in terms of number of packets. The value of a_{count} is user-definable and must be sufficiently large to differentiate misbehavior from normal packet loss rate. Lastly, S selects an initial packet sequence number a_{start}, indicating the sequence number of the packet where the audit begins. The source signs the audit request to enable the verification of its authenticity and integrity. In Figure 2, S selects node n_4 for audit, and sends the audit request through the disjoint path P_{Sn_4}.

4.1.2 Constructing a Behavioral Proof

When a node is audited, it constructs a behavioral proof of the set of all packets it forwards, from a_{start} to $a_{start} + a_{count}$, denoted by $X = \{x_1, x_2, \ldots, x_N\}$. Buffering packets themselves would require large amount of storage and significant overhead for transmission back to the source. On the other hand, Bloom filters [3] provide a compact representation of membership for a set $X = \{x_1, x_2, \ldots, x_N\}$ in an m-bit vector v with $m \ll N$. For an empty set X, all m bits of v are initialized to zero. A member x_i is added to Bloom filter X by passing x_i through k independent hash functions $h_l, 1 \leq l \leq k$ with each h_l having a range of $\{1, \ldots, m\}$. The corresponding bits $h_l(x_i), 1 \leq l \leq k$ of vector v are set to one. To check if y is a member of X, element y is hashed k times using h_l. If any corresponding bit location $h_l(y)$ in v is zero, element $y \notin X$. Else $y \in X$ with very high probability. Thus Bloom filters may yield a false positive, i.e., the filter may indicate $y \in X$ even though it is not. For perfectly random hash functions, the false positive probability p_f is given by [4]:

$$p_f = \left(1 - \left(1 - \frac{1}{m}\right)^{qN}\right)^q \approx \left(1 - e^{-\frac{qN}{m}}\right)^q \qquad (1)$$

In Figure 3(a), we show a Bloom filter v ($m = 10$) that has been initialized to zero, representing the membership set $X = \{\}$. Figure 3(b), shows element x_1 being added to v by passing through k independent hash functions $h_l, 1 \leq l \leq k$, with bits $h_l(x_i)$ of v set to 1, yielding membership set $X = \{x_1\}$. To check if y_1 is in X, y_1 is passed through h_l as in

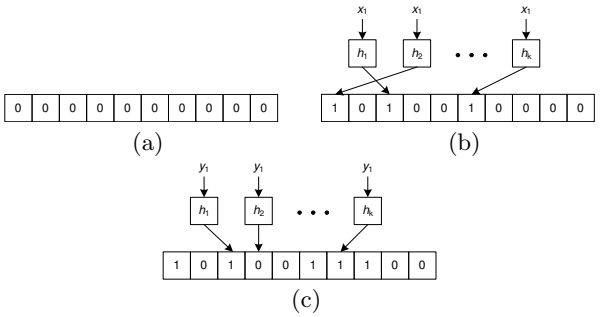

Figure 3: (a) Initialize Bloom Filter (b) x_1 is added to Bloom filter by passing through k hash functions with corresponding bits set to zero (c) Since $h_2(y_1)$ corresponds to a zero bit, y_1 not in Bloom filter.

Figure 3(c). Since $h_2(y_1)$ corresponds to a zero bit, $y_1 \notin X$.

The number of hash functions q that minimize the false positive probability p_f, is known to be $q = \ln_2\left(\frac{m}{N}\right)$, but any choice can be made to allow a graceful tradeoff between p_f and q. We can also compute the minimum storage required (size of vector v) so that $p_f \leq \epsilon$, to be equal to $m \geq N\frac{\log_2 \epsilon}{\ln 2}$.

Upon receiving an audit request, node n_i creates a proof of all packets it forwards to the next hop by constructing Bloom filter v_i. The audited node n_i inserts each packet $x_j, a_{start} \leq x_j \leq (a_{start} + a_{count})$ into his Bloom filter v_i. After a_{count} packets have been added to v_i, n_i signs v_i and sends it to S via the reverse path P_{n_iS}. Note that each Bloom filter is signed, acting as a public commitment to the packets forwarded that node. Misbehavior can be publicly verified via comparison with the source's Bloom filter.

In order to check n_i's audit (Bloom filter), the source constructs its own Bloom filter v_S in the same manner as n_i, i.e., all packets $x_j,\ a_{start} \leq j \leq (a_{start} + a_{count})$ are added to v_S. When S receives n_i's behavioral proof, it will then have two Bloom filters; v_S, which is guaranteed to correctly contain all packets in X, and v_i from node n_i.

4.1.3 Processing the Behavioral Proof

When S receives the behavioral proof from n_i, it verifies its authenticity and discards v_i if the signature check fails. If n_i fails to respond to the audit request, S may re-transmit the request using alternative paths. After a certain number of reply failures, S assumes that the node n_i is suspicious of misbehaving and continues with the algorithm execution.

The source performs a comparison of Bloom filters v_i, v_S by computing the inner product $\langle v_i, v_S \rangle$, which measures the similarity between vectors v_i, v_S. Let X_S denote the set of packets in v_S and X_i denote the set of packets in v_i. The magnitude of the inner product can be approximated by [4]:

$$\langle v_i, v_S \rangle \approx m\left(1 - \left(1 - \frac{1}{m}\right)^{q|X_i|} - \left(1 - \frac{1}{m}\right)^{q|X_S|} + \left(1 - \frac{1}{m}\right)^{q(|X_i|+|X_S|-|X_i \bigcap X_S|)}\right). \quad (2)$$

Given vector length m, cardinalities of X_i, X_S, and q hash functions, S can compute the size of the intersection set,

$$|X_i \bigcap X_S| \approx |X_i| + |X_S| - \frac{\log\left(\frac{<v_i, v_S>}{m} + \left(1 - \frac{1}{m}\right)^{q|X_i|} + \left(1 - \frac{1}{m}\right)^{q|X_S|}\right)}{q\log\left(1 - \frac{1}{m}\right)}. \quad (3)$$

The cardinality of $X_i \bigcap X_S$ provides a method to verify if packets in X_S are in X_i. Furthermore, S can maintain a sampling of X_S to perform membership tests on v_i for an additional verification of the packets in X_i. The sampling can be either random or packets of higher importance.

A node can arbitrarily construct its own Bloom filter to avoid any accusation of misbehavior by setting all bits of its filter to one. In such a case, $|X_i \bigcap X_S| \approx |X_S|$ since $X_S \subseteq X_i$ and any membership test would come out positive. However, S can easily verify if Bloom filter X_i contains packets not in X_i. The source can pick any $x \notin X_S$ and test if it is a member of X_i. If the membership test is positive, S can assume that $x \in X_i$ with a probability (1-p_f). The probability of false positive can be further reduced by repeating the experiment r number of times, yielding a successful identification of Bloom filter manipulation with a probability 1-$(p_f)^r$.

4.2 Search Phase - Single Misbehaving Node

So far we have illustrated how the source S evaluates the behavior of node n_i via auditing. We now show how S selects nodes for audit in order to identify misbehaving ones. We define the notion of a suspicious set $\mathcal{V}$ as the set of nodes $n_i \in P_{SD}$ which have not been shown honest. Initially, all nodes $n_i \in P_{SD}$ are placed in $\mathcal{V}$. When S audits node $n_i \in P_{SD}$, either: (a) $|X_i \bigcap X_S| \approx |X_S|$, or (b) $|X_i \bigcap X_S| \ll |X_S|$. For each case, the following conclusions can be drawn.

If n_i returns X_i: $|X_i \bigcap X_S| \approx |X_S|$, the source can conclude that all nodes upstream of n_i are honest. This is true, since if any node upstream of n_i was misbehaving, n_i would not receive packets in X_S and $|X_i \bigcap X_S| \ll |X_S|$. Therefore, S reduces the suspicious set to $\mathcal{V} = \{n_i, \ldots, n_k\}$. Note that n_i remains in $\mathcal{V}$, since it may correctly receive packets in X_S, construct v_i, but refuse to forward them to D.

If n_i returns X_i: $|X_i \bigcap X_S| \ll |X_S|$, the source can conclude that all downstream nodes are honest. This is true, since if any downstream node n_j is misbehaving, all nodes n_k upstream from n_j, including n_i, would return X_k: $|X_k \bigcap X_S| \approx |X_S|$, given that only one misbehaving node exists in P_{SD}. Since n_i returned X_i: $|X_i \bigcap X_S| \ll |X_S|$, the source concludes that the misbehaving node n_M must be upstream of n_i. Thus, S reduces the suspicious set to $\mathcal{V} = \{n_1, \ldots, n_i\}$.

By repeated audits, S reduces $\mathcal{V}$ to two nodes, node n_i that claims $|X_i \bigcap X_S| \approx |X_S|$ and node n_{i+1} that claims $|X_i \bigcap X_S| \ll |X_S|$. Hence, the search phase identifies the *misbehaving link*, i.e., the link in which packets are dropped. We now show how the misbehaving link is identified in cases of continuous and random packet dropping.

4.2.1 Case 1: Continuous Packet Dropping

We first consider the case where the misbehaving node drops all packets along P_{SD}. Let A denote a one-dimensional array of length $|P_{SD}|$ with $A[i] = |X_i \bigcap X_S|, 1 \leq i \leq |P_{SD}|$. Array A is almost sorted in descending order. This is a valid assumption since for any node upstream of the misbehaving node, $|X_i \bigcap X_S| \approx |X_S|$; and for any node downstream of the misbehaving node $|X_i \bigcap X_S| \ll |X_S|$. By converging on the

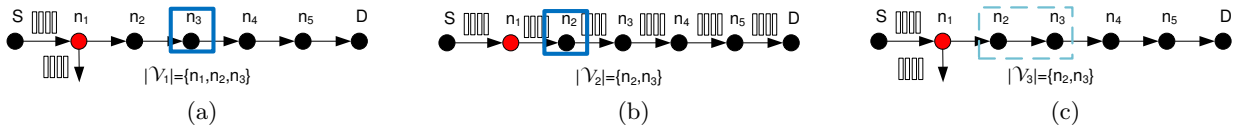

Figure 4: (a) S attempts to locate misbehaving node n_1 utilizing binary search. S audits n_3. Audit reply from n_3 fails membership test. (b) S audits n_2. Misbehaving node n_1 changes strategy and behaves honestly. Audit reply of n_2 passes the membership test. (c) Node n_3 successfully removed itself from suspicious set.

transition $A[i] \approx |X_S|$ and $A[i+1] \ll |X_S|$, the misbehaving link can be identified. This convergence can be achieved by performing binary search on A, requiring a maximum of $\log_2 |P_{SD}|$ steps. Let $k = |P_{SD}|$ and $\mathcal{V}_n$ be the suspicious set at iteration n, with $\mathcal{V}_0 = \{n_1, \ldots, n_k\}$. The source selects $n_i \in V_0$ such that $i = \lceil \frac{|\mathcal{V}_0|}{2} \rceil$. If n_i returns $X_i : |X_i \bigcap X_S| \approx |X_S|$, $\mathcal{V}_1 = \{n_i, \ldots, n_k\}$; else $\mathcal{V}_1 = \{n_1, \ldots, n_i\}$. Once $|\mathcal{V}| = 2$, the search has converged on the misbehaving link with termination time $\mathcal{O}(\log |P_{SD}|)$.

4.2.2 Case 2: Sophisticated Packet Dropping

We now consider a sophisticated misbehaving node that changes its dropping pattern to avoid identification. We first describe this behavior by an example. In Figure 4(a), misbehaving node n_1 drops packets. The source uses binary search to identify the misbehavior, choosing node n_3 to audit. The audit reply of n_3 fails the membership test, reducing the suspicious set to $\mathcal{V}_1 = \{n_1, \ldots, n_3\}$. The source then audits node n_2, as shown in Figure 4(b). Binary search is deterministic allowing n_1 to predict the order that nodes are audited. Node n_1 behaves honestly, thus n_2's audit response passes the membership test. By changing its behavior, n_1 removes himself from $\mathcal{V}$, as Figure 4(c) shows.

Deterministic search allows a misbehaving node n_M to "frame" a set of honest nodes. To mitigate the impact of the deterministic search, we utilize the following modifications: (a) we randomize the beginning a_{start} and duration a_{count} of each audit, so n_M cannot predict the timing of audits, and (b) we perform random binary search to prevent n_M from conjecturing the audit sequence. The modified scheme is shown in Algorithm 1.

Algorithm 1 Random Binary Search Audit Algorithm

```
Initialize : V_l ← n_1, V_r ← n_{|P_SD|}, V_n = {V_l, ..., V_r}
while |V_n| > 2 do
    audit(n_i) = V[rand]
    if |X_i ⋂ X_S| ≈ |X_S| then
        V_l ← n_i
    else
        V_r ← n_i
    end if
end while
return V_n
```

Random binary search converges in, on average, $\log_{\frac{3}{2}} |\mathcal{V}_0|$ iterations with termination time $\mathcal{O}(\log |P_{SD}|)$. To further prevent framing attacks, D sends an alert to S when the link's performance has returned to normal. Using these alert messages, S pauses the search and discards outstanding audits when no misbehavior is occurring. The search is resumed once S is alerted by D that misbehavior is occurring in P_{SD}. In Figure 4(b), if S pauses the search when there is no misbehavior, then the audit response of node n_2 would be discarded and node n_1 is unable to remove itself from $\mathcal{V}$.

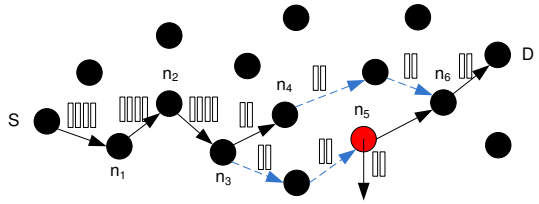

Figure 5: The search converges on link $n_4 - n_5$. S makes a slight alteration to P_{SD}, isolating n_4 and n_5, to determine that n_5 is the misbehaving node.

4.3 Identification Phase

Once the search process has converged on the misbehaving link, the two suspicious nodes n_i, n_{i+1} are excluded in turn from the routing path to the destination D. The node preceding the first suspicious node will split the traffic between n_i, n_{i+1} in turn. In Figure 5, S uses node n_3 to exclude in turn suspicious nodes n_4 and n_5. The source alerts D that two suspicious nodes are monitored via path exclusion. The destination creates two Bloom filters, $v_{D_i}, v_{D_{i+1}}$ corresponding to the packets routed through suspicious nodes n_i, n_{i+1}, and send them to S. The source compares v_i, v_{i+1} with its own filters $v_{S_i}, v_{S_{i+1}}$, and identifies the misbehaving node.

4.4 Multiple Misbehaving Nodes

We now examine the case of multiple independently misbehaving nodes. There exists two strategies for the nodes: (a) continuous misbehavior, and (b) randomly alter between honesty and misbehavior. In either case, we show S can identify, isolate, and locate the misbehaving nodes. The first step is to identify that more than one misbehaving node exists in P_{SD}, which is achieved as follows.

4.4.1 Case 1: Continuous Misbehavior

Assume $n_i, n_j \in P_{SD}$ are independently misbehaving and $n_i < n_j$, i.e., n_i is upstream of n_j. If n_i is misbehaving, then regardless of n_j's strategy, for all downstream nodes n_k, $|X_k \bigcap X_S| \ll |X_S|$ including n_j. Executing the search process of REAct will terminate with $|\mathcal{V}| = 2$, and S will make the path alterations to exclude in turn the two suspicious nodes. Note that the suspicious set $\mathcal{V} = \{n_i, n_{i+1}\}$, where one of n_i, n_{i+1} is the misbehaving node. Let that node be n_i. Since n_j is downstream from n_i, when S excludes n_i from the path, D will still report that both paths misbehave since both paths contain n_j. The only placement such that both paths do not contain n_j is for $\mathcal{V} = \{n_i, n_j\}$, in which case excluding in turn will cause misbehavior in each path. Thus, S identifies that multiple misbehaving nodes exist.

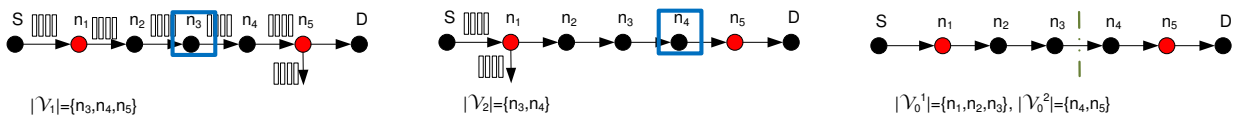

Figure 6: (a) n_5 misbehaves while n_3 is audited. $|X_3 \bigcap X_S| \approx |X_S|$. (b) Misbehavior strategies change (n_1 misbehaves, n_5 is honest). n_4 is audited and returns $|X_4 \bigcap X_S| \ll |X_S|$. Search converged to $|\mathcal{V}| \leq 2$ (c) Path is partitioned into two suspicious sets; $\mathcal{V}_0^1 = \{n_1, n_2, n_3\}, \mathcal{V}_0^2 = \{n_4, n_5\}$.

4.4.2 Case 2: Changing Strategies

In the case where multiple misbehaving nodes independently change their strategy from misbehaving to honesty, it is possible for these nodes to avoid detection. As an example, consider Figure 6(a) and assume n_1, n_5 are malicious. Node n_5 is misbehaving while n_1 faithfully forwards all packets. If the algorithm selects node n_3 for audit, the suspicious set will reduce to $\mathcal{V} = \{n_3, \ldots, n_5\}$. In Figure 6(b), the behavior pattern changes while searching with n_1 misbehaving and n_5 being honest. Thus, the search converges on $\mathcal{V} = \{n_3, n_4\}$, since any node downstream of n_3 has $|X_i \bigcap X_S| \ll |X_S|$. The problem arises because n_3 cannot change its response.

However, when excluding nodes in $\mathcal{V}$ in turn, both alternating paths contain misbehaving n_1. This results in both suspicious nodes misbehaving, indicating to S that P_{SD} contains multiple misbehaving nodes. If n_1, n_5 happen to change their strategy, both nodes appear honest. Hence, S always detects the existence of multiple misbehaving nodes. The randomization of the audit start and duration, in addition to the random search, prevents a misbehaving node from devising a strategy to frame honest nodes.

4.4.3 Isolation and Identification of Misbehavior

Once the source determines the existence of multiple misbehaving nodes, path P_{SD} is partitioned into two parts to isolate the misbehaving nodes from each other. The partition occurs between the two nodes in the suspicious set $\mathcal{V} = \{n_i, n_{i+1}\}$, i.e., P_{SD} is partitioned to P_{Sn_i}, $P_{n_{i+1}D}$. In Figure 6(c), S partitions $P_{SD} \rightarrow P_{Sn_2}, P_{n_3D}$.

With P_{SD} partitioned to $P_{Sn_i}, P_{n_{i+1}D}$, S executes REAct on each partition to locate the misbehaving node(s). In the single misbehaving node case, S uses alert messages from D to verify misbehavior in P_{SD} before accepting an audit response. Once P_{SD} is partitioned, S can only accept an audit response from $n_j \in P_{Sn_i}$ if misbehavior is occurring in P_{Sn_i}. The destination is unable to provide this, as D can only determine if misbehavior is in P_{SD}; not which partition.

Therefore, S simultaneously audits two nodes in each partition. In P_{Sn_i}, S audits a randomly selected $n_j \in P_{Sn_i}$ for the search, and node n_i. Auditing n_i determines if misbehavior is occurring in P_{Sn_i} by checking if X_i: $|X_i \bigcap X_S| \approx |X_S|$. Thus node n_i acts as a pseudo-destination. If $n_i = n_M$, it has only two strategies, (a) respond honestly with X_M: $|X_M \bigcap X_S| \approx |X_S|$, thus facilitating REAct or (b) lie and return X_M: $|X_M \bigcap X_S| \ll |X_S|$, in which case node n_{M+1} would return X_{M+1}: $|X_{M+1} \bigcap X_S| \approx |X_S|$, thus identifying the link $n_M - n_{M+1}$ has misbehaving.

Likewise on $P_{n_{i+1}D}$, S audits two nodes; a randomly selected $n_k \in P_{n_{i+1}}$ for the search, and node n_{i+1}. The audit response of n_{i+1} acts as a verification of how many packets from X_S have reached the partition, such that n_k is processed as $|X_k \bigcap X_{i+1}|$ to detect misbehavior. The source also checks if $|X_k \bigcap X_S| \leq |X_k \bigcap X_{i+1}|$, which prevents n_{i+1} from creating a Bloom filter which has only partial information. Node n_{i+1} therefore acts as a pseudo-source for $P_{n_{i+1}D}$ With these steps, REAct can be recursively executed to find multiple independently misbehaving nodes on P_{SD}.

5. PERFORMANCE EVALUATION

5.1 Simulation Setup

We randomly deployed 100 nodes within an 80×80 square area, and randomly selected source-destination pairs. For each pair we constructed the shortest path P_{SD} and randomly selected the set of nodes that misbehave. For each audited node n_i, we also constructed the shortest disjoint path P_{Sn_i} for sending the audit requests. We generated traffic from S to D following the UDP protocol. We assumed no packed loss or collisions on each network link. Each misbehaving node was simulating an either constant packet dropping pattern or a random strategy of rotating between honesty and misbehavior.

We compared the performance of REAct, with the performance of CONFIDANT [5] from the class of reputation-based systems and the performance of 2ACK [15] from the class of acknowledgment-based schemes. Specifically, we evaluated the communication overhead associated with the identification of compromised nodes and the incurred delay.

5.2 Communication Overhead

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 [9]. 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 $p = \{1, 0.5, 0.1\}$ [15]. For all three schemes we measured the communication overhead in terms of number of messages that need to be transmitted or overheard to perform neighbor monitoring.

Both CONFIDANT and 2ACK are proactive, incurring communication overhead regardless of the existence of misbehavior. However, REAct incurs overhead only if misbehavior exists, due to its reactive nature. To provide a fair comparison, we first considered the overhead of the proactive protocols for the duration of time required to identify the misbehaving node using REAct. The audit duration for REAct was set to 200 packets, i.e., each node had to provide proof for $a_{count} = 200$ packets every time it was audited.

5.2.1 Impact of path length

In Figure 7(a), we show the communication overhead in number of transmitted messages, as a function of path size

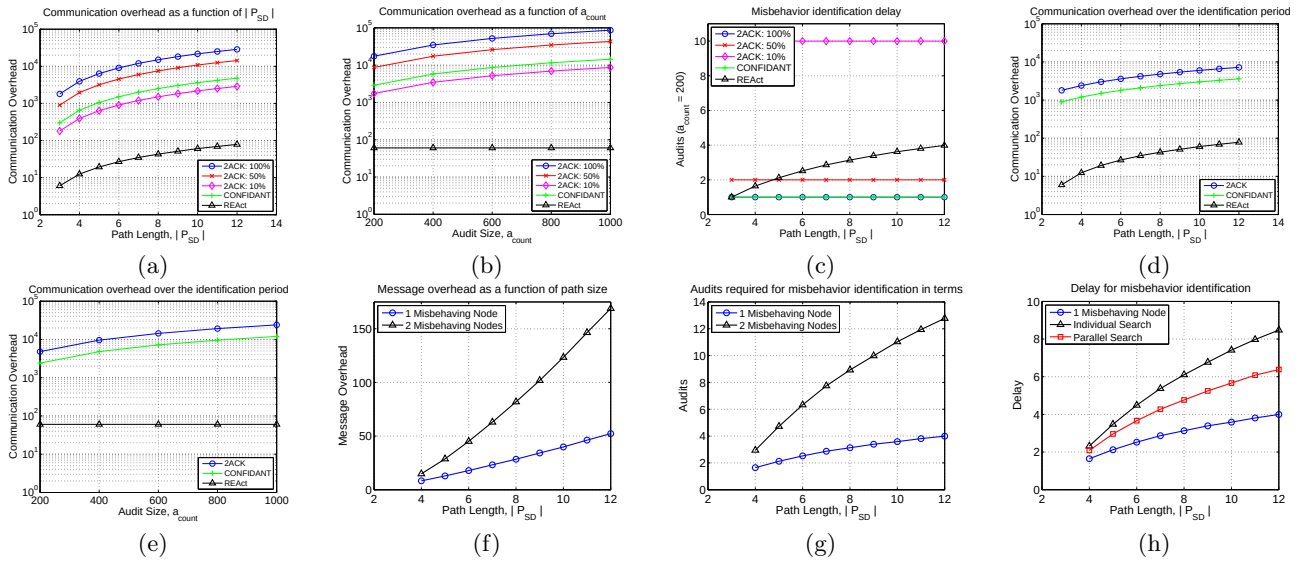

Figure 7: (a) Communication overhead as a function of path length for audit size of 200 packets. Overhead is computed as time required by REAct to converge on n_M. (b) Communication overhead as a function of audit size for $|P_{SD}| = 8$. (c) Identification delay as a function of path length in units of audit size. (d) Communication overhead as a function of path length for audit size of 200 packets. For each scheme, overhead is computed as time required to identify n_M. (e) Communication overhead as a function of audit size for $|P_{SD}| = 8$. (f) Communication overhead of REAct as a function of path size for two independently misbehaving nodes. (g) Number of audits required to identify the independently misbehaving nodes, as a function of path length. (h) Relative delay required to identify misbehaving nodes, as a function of path length.

$|P_{SD}|$. The Y axis is in logarithmic scale. We observe the communication overhead of REAct is almost 4 orders of magnitude less than the overhead of the proactive schemes. REAct only incurs overhead during the transmission of the Bloom filter to the source, whereas CONFIDANT and 2ACK actively monitor transmissions during the audit period. For all three schemes, the communication overhead increases almost linearly with path length. For CONFIDANT and 2ACK schemes, this linear increase is justified by the proportional increase of the number of transmissions that need to be monitored or acknowledged. For REAct, the linear behavior is justified by the proportional increase of path lengths from each audited node to the source.

5.2.2 *Impact of audit size*

The audit size parameter defines the number of packets needed to differentiate normal dropping rate from misbehavior. In Figure 7(b), we show the communication overhead as a function of audit size, a_{count}, for a path length of 8 nodes. Both proactive schemes incur a linear increase in communication overhead with audit size, since communication overhead incurs on a per-packet basis. The overhead for REAct depends on the number of audits, not the duration of each audit. Hence, it is independent of the audit size.

5.3 Identification Delay

While REAct provides significant savings in communication overhead, it requires a longer time to identify misbehavior, as multiple audits need to be performed. Proactive schemes require only a single audit duration to identify misbehavior since all nodes in path P_{SD} are monitored in parallel. Fortunately, the audits required by REAct grows logarithmically with path length due to the random binary search algorithm employed, resulting in fairly small increases in identification delay compared to savings in communication overhead.

In Figure 7(c), we show the identification delay for REAct, CONFIDANT, and 2ACK as a function of path length $|P_{SD}|$, in units of audits. We observe the logarithmic increase of identification delay with path length for the REAct scheme. CONFIDANT requires a single audit duration to identify misbehavior. 2ACK also requires a single audit when all packets are acknowledged. However, the identification delay increases when a fraction of the packets in one audit are acknowledged.

5.4 Identification Delay Based Comparison

According to Figure 7(c), the three compared schemes incur different delay in the misbehavior identification. We now evaluate the communication overhead incurred by each scheme, from the start of the node misbehavior until the identification of n_M. The communication overhead of CONFIDANT and 2ACK is measured for the duration of a single audit, while the overhead of REAct is measured for the logarithmic number of audits required to identify n_M.

In Figure 7(d), we show the communication overhead as a function of path length, for audit size of 200 packets. In Figure 7(e), we show the communication overhead as a function of audit size for path length of 8 nodes. Even in the case where the communication overhead is measured only during the identification time, REAct significantly outperforms the proactive schemes. Both CONFIDANT and 2ACK are sensitive to path length and audit size, leading to very high energy and bandwidth expenditure for misbehavior detection, while REAct allows for a graceful tradeoff between communication overhead and delay in misbehavior identification.

5.5 Multiple Misbehaving Nodes

We also evaluated the performance of REAct when P_{SD} contains two independently misbehaving nodes which randomly change from misbehavior to normal behavior. To determine the behavioral pattern for each n_M, time was divided in epochs. At each epoch, each n_M would decide whether to behave or misbehave according to the outcome of a fair coin flip. The duration of each epoch was randomly selected from the range of 1 to 400 packets.

In Figure 7(f), we show the communication overhead of REAct as a function of path length for an $a_{count} = 200$. The single misbehaving node case is depicted as reference. We observe the communication overhead grows up to three times larger compared to the single misbehaving node case, due in part to overhead incurred until the source realizes multiple misbehaving nodes exist.

In Figure 7(g), we show the audits required for identification of the misbehaving nodes, as a function of path length. Again, the single misbehaving node case is plotted for reference. We observe that multiple misbehaving nodes significantly increases the number of audits required, when path length grows large. Note that in the two misbehaving node scenario, the number of audits does not directly correspond to delay, as multiple audits are performed in parallel.

In Figure 7(h), we show the relative delay required for misbehavior identification, as a function of path length. A comparison is shown between the source searching both path partitions in parallel, and searching the partitions in turn. Parallel search is possible if the misbehaving nodes simultaneously misbehaving. The space between the parallel search line and the individual search line gives an expected delay.

6. CONCLUSION

We studied the problem of routing misbehavior in wireless ad hoc networks. Specifically, we addressed the problem of identifying misbehaving nodes that refuse to forward packets to the destination. We proposed a reactive identification scheme called REAct which relies on the random audit of a subset of nodes along the source/destination path to identify misbehaving nodes. Each audited node uses Bloom filters to construct a storage and communication-efficient behavioral proof of the packets it forwarded. We showed that REAct significantly reduces the communication overhead associated with the misbehavior identification process compared to reputation-based and acknowledgment-based schemes. This reduction in resource expenditure comes at the expense of a logarithmic increase in the identification delay, due to the reactive nature of our scheme.

7. REFERENCES

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

[2] K. Balakrishnan, J. Deng, and P. K. Varshney. Twoack: Preventing selfishness in mobile ad hoc networks. In *WCNC 2005*, 2005.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004.

[5] S. Buchegger and J.-Y. L. Boudec. Performance analysis of the confidant protocol (cooperation of nodes: Fairness in dynamic ad-hoc networks). In *MobiHOC 2002*, June 2002.

[6] S. Buchegger and J.-Y. L. Boudec. Self-policing mobile ad-hoc networks by reputation systems. *IEEE Communications Magazine*, pages 101–107, 2005.

[7] L. Buttyan and J.-P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM/ Kluwer Mobile Networks and Applications*, 8(5), 2003.

[8] J. Dyer, M. Lindemann, R. Perez, R. Sailer, and L. van Doorn. Building the ibm 4758 secure coprocessor. *IEEE Computer*, 34:57–66, October 2001.

[9] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *INFOCOM 2001*.

[10] V. Gligor. Handling new adversaries in secure mobile ad-hoc networks. In *ESNS 2007*, 2007.

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

[12] M. Jakobsson, J.-P. Hubaux, and L. Buttyan. A micropayment scheme encouraging collaboration in multi-hop cellular networks. In *Financial Crypto*, 2003.

[13] 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.

[14] A. Liu and P. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *IPSN 2008*. SPOTS Track, 2008.

[15] 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, May 2006.

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

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

[18] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. Tcp selective acknowledgment options. *RFC 2018*.

[19] P. Michiardi and R. Molva. Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *CMS 2002*, 2002.

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

[21] K. Paul and D. Westhoff. Context aware detection of selfish nodes in dsr based ad-hoc networks. In *IEEE Globecom*, 2002.

[22] 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.

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