RESOURCE-EFFICIENT MISBEHAVIOR IDENTIFICATION IN WIRELESS AD HOC NETWORKS

by

William David Kozma Jr

A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

MASTERS OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2009

STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: _____
William David Kozma Jr

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

_____ _____
Loukas Lazos
Assistant Professor

Date

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my academic advisor Dr. Loukas Lazos. Your passion for your work encouraged me to pursue my own goals, while your attention to details instilled in me a unique perspective on which to approach problems. I am deeply grateful for your taking me on as a student. Our academic work together has been a truly rewarding and enriching experience, while the lessons learned will help guide me throughout the course of my life in whatever avenue I pursue.

I would also like to thank the members of my defense committee, Dr. Marwan Krunz and Dr. Srini Ramasubramanian, for both supporting my degree goals and the valuable time spent in the classroom.

Lastly I would like to thank my family. To my wife, Kimberly, who has supported my ambitions from the very beginning, encouraging me to push my limits while doing all she can to help along the way. To my brothers, who provide me with that quite strength. And to my parents, you have provided everything that a son could every ask for, and so much more.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF FIGURES – *Continued*

LIST OF FIGURES – *Continued*

ABSTRACT

Wireless ad hoc networks rely on multi-hop routes to transport data from a source to a destination. Each node is responsible for relaying traffic towards the destination, thus implementing the routing function in a collaborative manner. However, the easy access of an increasingly sophisticated user pool 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 take advantage of the multi-hop routes provided by cooperative nodes while refusing to forward packets itself, in order to conserve its energy resources (selfishness), or simply degrade the network performance (maliciousness).

In this thesis, we investigate the problem of uniquely identifying the set of misbehaving nodes on which packet forwarding fails. We map the misbehavior identification problem to the classic game of 20 questions and different versions of Rényi-Ulam games. The advantage of our approach is that the behavior of each node is evaluated on a per-packet basis, without incurring per-packet overhead. Utilizing this mapping, we develop REAct, a resource-efficient misbehavior identification scheme. Our scheme relies on the source auditing nodes in a resource-efficient and publicly verifiable manner. Using Rényi-Ulam inspired auditing strategies, we show that a source can identify any number of misbehaving nodes, given sufficient audits. Additionally, our mapping to Rényi-Ulam games allows REAct to implicitly assume the existence of collusion among misbehaving nodes. We explore the tradeoff between the delay in misbehavior identification and the associated communication overhead.

CHAPTER 1

Introduction

1.1 Motivation and Scope

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 in a variety of environments, ad hoc networks find numerous civilian applications, such as collaborative computing [35], emergency services [46], vehicular networks [70, 71], patient monitoring [43] and environmental monitoring [42], as well as military applications, such as area surveillance [24], and target tracking [7].

Without the support from a backbone infrastructure, ad hoc networks rely on the collaborative implementation of network functionals. All nodes participate in the network by sharing their individual resources for the purpose of realizing network-wide services. As an example, consider Figure 1.1 depicting an ad hoc network of 12 nodes. A source S wants to establish communication with a destination D. Due to the wireless constraints on their radios, S and D are unable to communicate directly. Thus, the source uses a multi-hop path to route data to the destination.

This collaborative communication model assumes that a path of intermediate nodes from S to D can be established and that these nodes are willing to forward data on behalf of the source. Any collaborative network function such as multi-hop routing is based upon the assumption that network nodes comply with the protocols that specify the collaboration.

However, an increasingly sophisticated base of malicious users has gained access to off-the-shelf networking devices. Such users may modify the software or hard-

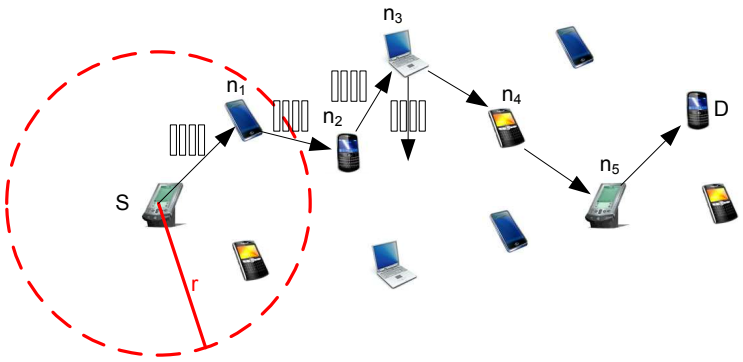


Figure 1.1: An ad hoc network of 20 nodes. The circle with radius r represents the communication range of the nodes radio. The source S uses a multi-hop route to send data to the destination D. Node n_3 does not comply with the routing protocol specifications and drops packets destined to D.

ware of their devices in order to maximize their individual network benefit, while expending a minimum amount of resources. Moreover, malicious users may attempt to degrade the network performance through protocol misbehavior. Note that at present, it is too expensive to equip every network device with tamper-proof hardware [19], thus sophisticated users can easily gain access to both the software and hardware modules of the devices.

In this thesis, we consider 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, either to degrade throughput or conserve their energy resources.* In Figure 1.1, malicious node n_3 attempts to conserve its own energy by refusing to forward packets towards the destination. Such misbehavior has been shown to have a severe impact on the network throughput [8, 9, 40, 41, 44]. We focus on identifying n_3 in a publicly verifiable and resource-efficient manner.

Currently proposed solutions to the misbehavior problem attempt to either identify and revoke the misbehaving nodes [8, 9, 10, 40, 41, 44], or provide incentives for cooperation within the network [12, 13, 14, 15, 30, 62, 72]. In incentive-based approaches, referred to as credit-based systems, nodes obtain a form of credit for

faithfully forwarding packets to the next hop. Nodes can later use their accumulated credit to have their own packets forwarded by other nodes. Thus nodes are motivated to cooperate in packet forwarding in order to be able to communicate in the network. However credit-based systems do not address the case of malicious users who aim at network disruption, and have no motivation in credit accumulation for transmitting their own packets.

Misbehavior identification approaches can be classified to reputation-based systems [8, 9, 10, 11, 21, 24, 32, 44, 45, 50, 59, 63], and acknowledgment-based systems [2, 3, 40, 48, 68]. In reputation-based systems, nodes monitor the behavior of their neighbors by overhearing that a received packet was faithfully forwarded to the next hop. However, this requires operation in promiscuous mode which can be almost as expensive as transmission [20]. Note that monitoring nodes have to expend energy to overhear every packet received and forwarded by the monitored node. Furthermore, neighborhood monitoring mechanisms are difficult to implement in multi-channel networks where nodes may be engaged in parallel transmissions in orthogonal frequency bands.

Acknowledgment-based systems rely on the explicit acknowledgment of each transmitted message. Every packet received must be acknowledged two or more hops upstream, thus verifying that intermediate nodes cooperated in packet forwarding. Sending acknowledgments for every packet adds significant communication and energy overhead in the misbehavior identification process. Both reputation-based and acknowledgment-based systems are proactive in nature, and thus continuously monitor the nodes' behavior.

We address the problem of identifying the misbehaving node with resource efficiency in mind. This is accomplished by developing a method in which the source can perform per-packet evaluation of packets forwarded by an individual node, without incurring per-packet overhead.

1.2 Main Contributions and Thesis Organization

The main contribution of this thesis is the development of a *reactive* misbehavior identification scheme that does not rely on the per-packet monitoring of nodes. To achieve this goal, we map the problem of misbehavior identification to the classic Rényi-Ulam game of 20 questions [60, 67]. Rényi-Ulam games have been extensively used in various contexts including error correction codes [4], selecting, sorting, and searching in the presence of errors [51, 61, 64], to name a few. They provide efficient searching strategies in the presence of errors. We develop three communication-efficient algorithms for locating the misbehaving nodes, generally referred to as REAct, based on different versions of Rényi-Ulam games. Our mapping allows the per-packet evaluation of the behavior of nodes without incurring the per-packet communication overhead. The source can uniquely identify the misbehaving nodes, with communication overhead that grows logarithmically with the path size. Our formulation copes with colluding adversaries who coordinate their behavioral patterns to avoid identification and frame honest nodes. Collusion is not explicitly addressed in most prior work [2, 3, 9, 40, 44]. REAct achieves at least an order of magnitude savings in communication overhead when compared to other schemes. This improvement comes at the slight increase in the delay required to identify the misbehaving node.

The remainder of the thesis is organized as follows. In Chapter 2, we describe different types of routing misbehavior in wireless ad hoc networks. We then discuss related work to the misbehavior identification problem. Finally, we formally define our problem and state our network and adversarial models. In Chapter 3, we present a simple version of our reactive misbehavior identification algorithm, thus establishing the core ideas of our approach. In Chapter 4, we introduce Rényi-Ulam games and their real-world applications. We then show how the misbehavior identification problem can be mapped to Rényi-Ulam games. Inspired by this analogy, we develop

REAct, our resource-efficient misbehavior identification scheme. In Chapter 5, we evaluate the performance of REAct comparative to previously proposed schemes. In Chapter 6, we present our conclusions.

CHAPTER 2

Misbehavior in Wireless Ad Hoc Networks

2.1 Introduction

Ad hoc networks are based on the assumption that all nodes collaborate to realize network-wide services. However, this implicit trust placed in network nodes is not always preserved. A misbehaving node can act in a selfish manner, in order to conserve its own resources (such as energy), thus refusing to relay packets to the next hop. Moreover, a malicious node can degrade network performance by dropping all packets routed through it. Regardless of its motive, a misbehaving node violates the core ad hoc network principle of collaboration. Node misbehavior in the routing function has been shown to have a severe impact on the network throughput [8, 9, 40, 41, 44].

Several attacks have been demonstrated against routing protocols in ad hoc networks [8, 9, 16, 25, 28, 38, 40, 44, 47, 56, 57]. In the sinkhole attack, a misbehaving node attempts to attract traffic by falsely advertising a shortest route to multiple destinations [16, 47, 56]. Thus, neighboring nodes route their traffic through the misbehaving node, allowing it to drop/modify/analyze a large volume of packets. Networks in which all nodes periodically transmit data to a sink, such as a monitoring network, are particularly vulnerable to the sinkhole attack since data packets have a single destination.

In the the wormhole attack [25, 38, 57], two (or more) nodes establish a low-latency link between distant parts of the network. Messages received at one end of the wormhole are transmitted back on the other end. Since the wormhole is a low-latency link, nodes on either side of the wormhole will appear as neighbors, and

eventually all packets destined from one side of the network to the other will traverse the wormhole. Notice that in this attack, the misbehaving nodes are not required to broadcast messages advertising the low-latency link. This occurs naturally during route discovery since all received packets are rebroadcast through the wormhole. Once established, the misbehaving nodes can launch additional attacks on the network based on the large volume of traffic traversing the wormhole. Preventative measures against the wormhole attack are based on topological consistency checks [25, 27, 38].

On-demand routing protocols for ad hoc networks have been shown vulnerable to the rushing attack [28]. In this attack, the misbehaving node tampers with the route request packet and modifies the routing path. Once modified, the node rushes the packet to the next hop. Since under most protocols only the first copy of the route request is accepted (all subsequent ones are dropped), the misbehaving node attempts to forward the modified route request before any others are transmitted, thus causing the modified packet to be used for the duration of the route discovery and establish a false routing path.

In the blackhole attack [37, 65], the misbehaving node advertises the shortest path to a particular destination node whose traffic it wants to intercept. Once a route request is received by the misbehaving node, it immediately replies to the source. Since most route discovery processes accept the first route discovered, the traffic is routed through the misbehaving node.

The most common form of misbehavior is packet dropping [8, 9, 40, 44]. In this attack, the misbehaving node participates in the routing path establishment process. Once the routing path is established, the misbehaving node simply refuses to forward packets to the next hop. Although a source and destination may employ an end-to-end acknowledgment scheme, such as the one used in the TCP protocol, an acknowledgment scheme may not be employed on a per-link basis. The source and destination can recognize that a performance drop has occurred on the routing

path, but are unable to determine the problematic link. This thesis focuses on this last type of misbehavior in which the source and destination attempt to determine the node(s) that drop packets along the routing path. We now describe related work with respect to the misbehavior identification problem

2.2 Related Work

Previously proposed methods for addressing the misbehavior problem can be classified into, (a) credit-based systems [12, 13, 14, 15, 30, 62, 72], (b) reputation-based systems [8, 9, 10, 11, 21, 24, 32, 44, 45, 50, 59, 63], and (c) acknowledgment-based systems [2, 3, 40, 48, 68].

2.2.1 Credit-Based Systems

Credit-based systems [12, 13, 14, 15, 30, 62, 72] are designed to provide incentives for forwarding packets. Buttyan and Hubaux [12, 13, 14] proposed a system in which nodes receive credit for each packet they forward, and spend their accumulated credit to transmit their own packets. This is accomplished through the use of a counter called the *nuglet counter*. The nuglet counter is incremented each time the node forwards a packet, and decremented each time the node transmits its own packet. The nuglet counter cannot take on a negative value and cannot be arbitrarily changed by the node. To enforce this rule, the nuglet counter is implemented in a tamper-proof hardware module, called the *security module*. The security module is assumed to provide universal protection from both software and physical attacks.

Zhong et al. [72] proposed Sprite, in which nodes collect *receipts* for the packets that they forward to other nodes. For a packet sent from a source to a destination, each node along the path records a hash of the packet as the receipt, and forwards the packet to its next hop. When the node has a high-speed link to a Credit Clearance Service (CCS), it uploads its receipts. The CCS determines the value of the receipts

and provides credit in exchange. Credit is only granted if the destination reports a receipt verifying reception of the packet and if the node was on the routing path. Once verified, credit is removed from the sources account and given to each node who participated in packet forwarding. Thus nodes that transmit their own packets but do not cooperate in packet forwarded will incur a debt at the CSS and can be identified.

Crowcroft et al. [15] propose a scheme which not only rewards nodes for participating in packet forwarding with credit, but takes into account congestion and traffic flow. When sending a packet, the source computes a congestion price, which is a metric defined by the required power for transmission and the available bandwidth. It then compares this price to its personal willingness-to-pay parameter, which the source continually adjusts based on its personal observations. By taking into consideration bandwidth in computing the cost (credit) required to send a message to the destination, the scheme avoids overwhelming low cost routes, as they would increase in costs as they become saturated. Power and bandwidth metrics are dynamically updated based on shared information among nodes.

Salem et al. [62] proposed a scheme to provide incentives to nodes in multihop cellular networks. The scheme relies on the fact that all network traffic must travel through the base stations (i.e. cell towers), and that all base stations are owned by a single trusted operator. When the source sends a packet, it appends a keyed hash of the entire packet. Each intermediate node re-hashes the entire packet, including the previously appended hash. The previous nodes hash is then replaced with the new intermediate nodes hash. Once at the base station, the hash is verified and the packet is transmitted over the backbone network, where it is re-transmitted to the destination from a nearby base station. The source is charged immediately by the base station upon receipt of a packet, while the destination is charged a small amount when the packet is re-transmitted. This amount is refunded once the destination acknowledges the reception of the packet, thus preventing the destination

from cheating the system by claiming packets were never received.

While credit-based systems motivate selfish nodes to cooperate in packet forwarding, they provide no incentive to malicious nodes that target the network throughput. Such nodes have no incentive to collect credit and receive no punishment for non-cooperation. Furthermore, tamper-proof hardware [22] is currently too expensive to integrate in every network device, while providing an unverifiable level of security [1]. Sprite removes this requirement, at the expense of requiring the presence of a CCS. Lastly, credit-based systems lack a mechanism for identifying the misbehaving node(s), allowing them to remain within the network indefinitely.

Huang et al. [29] even go so far as to question whether or not there exists a need for credit-based schemes. Some of the issues they bring to light is that all network nodes are not treated fairly. Nodes located on the outskirts of the network will encounter less traffic to be forwarded, thus accumulating less credit compared to nodes located in the center of the network. Thus there exists an inherent disadvantage to a subset of nodes. They also claim that to be effective, a credit-based system must be uniquely designed for a given network, which conflicts with it idea of ad hoc networks and their corresponding technologies.

2.2.2 Reputation-Based Systems

Reputation-based systems [8, 9, 10, 11, 21, 24, 32, 44, 45, 50, 59, 63] use neighborhood monitoring techniques to identify misbehaving nodes. Marti et al. [44] proposed a scheme which relies on two modules, the *watchdog* and the *pathrater*. The watchdog module monitors the behavior of their next hop node by operating their radio in promiscuous mode. Once a node forwards a packet to the next hop, the node overhears to verify that the next hop node faithfully forwarded the packet. The scheme is based on the assumption that links between nodes are bi-direction and nodes utilize omni-direction antennas. A cache is used to store packets that wait for verification. If packets remain in the cache longer than a threshold period,

the watchdog makes an accusation of misbehavior. The pathrater module uses the accusations generated to chose a path that will most likely avoid misbehaving nodes.

Buchegger and Le Boudec [8, 9, 10] proposed a scheme called *CONFIDANT*, which is built upon the watchdog/pathrater model. Nodes perform neighborhood monitoring using their radios in promiscuous mode while selecting paths that attempt to avoid misbehaving nodes. Whereas Marti et al. proposed using only the previous hop for monitoring, CONFIDANT requires all neighboring nodes to operate in promiscuous mode for monitoring, thus replying on a neighborhood watch. In addition, monitoring nodes notify other nodes of detected misbehavior through the broadcast of alarm messages. Instead of including a proof of the misbehavior in the alarm message, a scheme based on Pretty Good Privacy (PGP) [73] is implemented to determine the trust level of the alarm message.

Soltanali et al. [63] propose a reputation-based scheme consisting of four modules: a Monitor, a Opinion Manager, a Reputation Manager, and a Routing/Forwarding Manager. The Monitor module monitors the nodes neighbors via the watchdog model, verifying that neighboring nodes faithfully participate in packet forwarding. Based on observations from the Monitor, the Opinion Manager formulates opinions of the nodes behavior and periodically advertises them to neighboring nodes. The Reputation Manager accepts these opinions and processes them to arrive at a trust metric for a specific node. When establishing a routing path to a destination, the Routing/Forwarding Manager uses these trust metrics to avoid including untrustworthy (misbehaving) nodes.

Ganeriwal and Srivastava [21] use a Bayesian model to map binary ratings to reputation metrics, using a beta probability density function. Each sensor computes a reputation ranking for its neighbors, defining them as cooperative or noncooperative. The ranking is based on multiple factors, including but not limited to routing consistency and packet integrity. Nodes can also share information regarding their classification of neighbors as cooperative/noncooperative. Jøsang and Ismail [32]

presents similar work on how to derive reputation rankings using beta probability functions based on feedback of neighboring node behavior. Likewise, Buchegger and Le Boudec [11] investigate the effects of rumor spreading in ad hoc networks and propose a reputation-based scheme based on a Bayesian model. They also attempt to identify lies and exclude them as input to their reputation model.

He et al. [24] proposed SORI, which monitors neighboring nodes using a watchdog mechanism and propagates this information to nearby nodes, thus relying on both first- and second-hand information. Each node monitors all neighboring nodes, while maintaining a neighborhood list. The neighborhood list contains the number of packets each neighbor received and the number forwarded. Periodically, neighboring nodes exchange reputation information. This second-hand information is added to the nodes observations to compute an overall evaluation record for a node. SORI takes the additional step of punishing nodes deemed to be misbehaving. Neighbors of a misbehaving node will probabilistically drop its packets, thus encouraging cooperation among nodes. SORI includes a mechanism to prevent retaliation attacks in which nodes continually increase the probability of dropping each others packets. The authors address security issues such as node impersonation by requiring the use of an authentication mechanism based on one-way hash chains.

Rebahi et al. [59] proposed a reputation-based scheme which also relies on first- and second-hand information. However the authors propose two different methods for nodes to acquire the second-hand information, i.e., the reputation information originating from neighboring nodes. In the first method, as soon as a node witnesses misbehavior, defined according to a threshold number of packet drops, the node immediately broadcasts the accusation. Thus the proactive transmitting of reputation information allows all nodes in the network to have up-to-date behavioral information about their neighbors. However, since the proactive broadcasting of information may require unacceptable bandwidth requirements, thus diminishing the networks functionality, nodes can also acquire second-hand information in

an on demand manner. In much the same way that on demand routing protocols request route information, a node transmits a packet to the network requesting reputation information from other nodes. Thus network resources are only consumed to transfer reputation information that is requested.

Michiardi and Molva [45] proposed CORE, in which nodes create a composite reputation rating for a given node by combining the nodes subjective reputation, its indirect reputation and its functional reputation. The subjective reputation is calculated from direct observation of the nodes behavior, using a weighted average of both current and past observations. The indirect reputation is a value calculated based on second-hand observations made by other nodes in the network. A nodes functional reputation is based on task-specific behavior. Thus it is computed based on its reputation in packet forwarding, routing, etc. Denial-of-service attacks based on misbehaving nodes broadcasting negative ratings for honest nodes are prevented by preventing nodes from broadcasting negative behavior. Thus when sharing reputation metrics, node are restricted to sharing only positive ratings.

Paul and Westhoff [50] proposed a scheme which can identify different types of misbehavior through routing message verification and packet comparisons. In particular, they focus on securing DSR to attacks, in which a misbehaving node either (a) refuse to forward route request packets, (b) forwards route requests without adding itself to the routing path, or (c) adds unrelated nodes to the route request. The scheme verifies routing messages through the use of an un-keyed hash chain, while nodes compare RREQ headers to a local cache consisting of headers from overheard packets to identify misbehavior. Each intermediate node along the path thus monitors its neighboring nodes, and send any accusations of misbehavior to the source, along with the type of misbehavior they witnessed. The source analyzes all accusations received, and takes action based on the type of misbehavior witnessed.

The process of node monitoring becomes complex in the case 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 being unable to monitor their peers. Moreover, operating in promiscuous mode requires up to 0.5 times the amount of energy for transmitting a message [20], thus making message overhearing an energy expensive operation. Finally, reputation-based systems are proactive in nature, requiring the constant monitoring of nearby nodes. Hence, overhead is incurred on all nodes regardless of whether a misbehaving node exists in a neighborhood.

2.2.3 Acknowledgment-Based Systems

Acknowledgment-based systems [2, 3, 40, 48, 68] 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 (TWOACK) to verify cooperation. For every packet a node receives, it sends a TWOACK along the reverse path, verifying to the node 2-hops upstream that the intermediate node faithfully cooperated in packet forwarding. Packets that have not yet been verified 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. [40] improved on TWOACK by proposing 2ACK. Similar to TWOACK, nodes explicitly send 2-hop acknowledgments (2ACK) to verify cooperation. To reduce overhead, 2ACK allows for only a percentage of packets received to be acknowledged. Additionally, 2ACK uses a one-way hash chain to allow nodes in the routing path to verify the origin of packets they are acknowledging, thus preventing attacks in which a misbehaving node drops the original packet and forwards a spoofed packet.

Padmanabhan and Simon [48] proposed a method called *secure traceroute* to identify the link on which misbehavior is occurring. Instead of the standard traceroute operation, which relies on nodes responding to expired packets, secure tracer-

oute verifies the origin of responses and uses traceroute packets that are indistinguishable from data packets. Secure traceroute proceeds hop by hop, although instead of responding to expired packets, the source establishes a shared key with the node. By encrypting the packets, secure traceroute packets are indistinguishable from data packets and cannot be selectively dropped. A Message Authentication Code (MAC) is utilized for authenticating the packets origin. Although traceroute is considered a reactive approach, secure traceroute is proactive, requiring connected nodes to transmit "keep-alive" packets when they have node data of their own to send, albeit at lower data rate.

Xue and Nahrstedt [68] proposed the Best-effort Fault-Tolerant Routing (BFTR) scheme, which relies on end-to-end acknowledgment messages to monitor packet delivery ratio and select routing paths which avoid misbehaving nodes. Similar to the DSR routing protocol, the source floods RREQ messages to discover a routing path to a destination. However, RREP packets must be sent along the reverse path and must be signed with a shared secret key between the source and destination. Also, the destination responds to multiple RREQ, thus providing the source with multiple paths to choose from. The source selects the shortest path for packet routing. During transmission to the destination, the source monitors the feasibility of the routing path, based on the end-to-end acknowledgments sent by the destination. Using a proposed heuristic, the source varies the routing path to maintain feasibility. Thus, the goal of BFTR is to avoid misbehaving nodes.

Awerbuch et al. [2] proposed an on demand routing protocol that probes the path to identify the faulty link. Once misbehavior is identified as occurring, the source begins probing nodes on the routing path by asking nodes to acknowledge all packets received. Probing is performed according to a binary search, in which the binary response of probed nodes are {failed, successful}. Once the faulty link has been identified, a weight metric is utilized to increase the value of the faulty link, thus avoiding including it in future routing paths. To avoid a misbehaving

node from dropping the acknowledgments of probed nodes, the acknowledgment are attached to packets from previous nodes such that the misbehaving node cannot drop only a subset of acknowledgment messages. The source makes no attempt to identify the individual node(s) causing the misbehavior.

Acknowledgment-based systems 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. Awerbuch et al. further reduces overhead through its on demand characteristic, however it only identifies the faulty link, thus failing to identify the node causing the misbehavior.

2.3 Models and Problem Statement

2.3.1 Network Model

We consider a multi-hop ad hoc network where nodes collaboratively relay traffic according to an underlying routing protocol such as DSR [31] or AODV [54]. The path P_{SD} used to route traffic from a source S to a destination D is assumed to be known to S. This is true for source routing protocols such as DSR where the entire P_{SD} is included in every packet. If DSR is not used, the source can identify the nodes in P_{SD} by performing a traceroute operation. For simplicity, we number nodes $\{n_1, \ldots, n_k\}, k = |P_{SD}|$, located in a path P_{SD} in an ascending order, i.e., node n_i is *upstream* of n_j if $i < j$ and is *downstream* of n_j if $i > j$.

We assume that the source and the destination collaboratively monitor the performance of P_{SD}. The destination is responsible for periodically reporting to the source critical metrics such as throughput or delay. Periodic updates with average performance metrics are sufficient for our purpose. If a misbehaving node drops the periodic updates as part of it's misbehavior pattern, the source interprets the lack of updates as a sign of misbehavior in P_{SD}. Likewise the destination explicitly

alerts the source in case the performance in P_{SD} is restored. This explicit alert is used to pause the misbehavior identification process and account for: (a) temporal variations of performance due to traffic or intermittent connectivity, and (b) random behavioral patterns of the misbehaving nodes.

We assume the network is k-connected, with $k-1$ representing the maximum number of colluding misbehaving nodes. This is required to prevent colluding nodes from framing an honest node of misbehavior during the identification process. Additionally, we initially consider a quasi-static network in which the path P_{SD} does not change for the entire duration of the misbehavior identification process. We later relax this assumption, allowing changes in the path P_{SD} due to node mobility. Note that we do not require the existence of an end-to-end per packet acknowledgment mechanism.

We assume that the integrity, authenticity, and freshness of critical control messages can be verified using resource-efficient cryptographic methods. For example, a public key cryptosystem realized via computationally-efficient elliptic curve cryptography may be used to verify the authenticity and integrity of messages while providing confidentiality [39]. Nodes audited by the source use their private key to sign their commitment of the set of packets forwarded to the next hop. This signed commitment becomes a publicly verifiable behavioral proof, as any node in the network can verify the commitment of packets forwarded by using the corresponding public key. Likewise, due to the secret nature of the private keys, all nodes are proven of the origin of the behavioral claim without requiring contact with the origin node. 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 [18, 58, 69]. Alternatively, methods based on symmetric keys can be used to protect critical messages [26, 49, 55].

2.3.2 Adversarial Model

We assume that a set M of misbehaving nodes exist in a path of length $k \geq |M|$. Misbehaving nodes can be located anywhere in P_{SD}. The source and destination have a mutual interest in communicating or framing honest nodes. Misbehavior is not in the form of packet dropping, since either the source or destination would have refused to establish the connection. Hence we consider the misbehavior of the source and destination separately. Following Kerckhoff's principle, misbehaving nodes are assumed to be aware of the routing protocol and the details of the mechanism used for misbehavior identification. The goal of misbehavior is twofold: (a) degrade throughput between the source and and destination, and (b) remain undetected. We consider two models with respect to the behavioral pattern of nodes in M.

Independently misbehaving nodes: In this model, nodes in P_{SD} misbehave independently without coordinating their packet dropping patterns. Misbehavior is modeled after an ON/OFF process in which nodes alternate between dropping packets and behaving normally. The duration of the misbehaving/behaving period is exponentially distributed with parameters μ_1, μ_2.

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

2.3.3 Problem Statement

Consider a path P_{SD} which contains a set M of misbehaving nodes. Nodes in M misbehave according to the adversarial models defined in Section 2.3.2. *We*

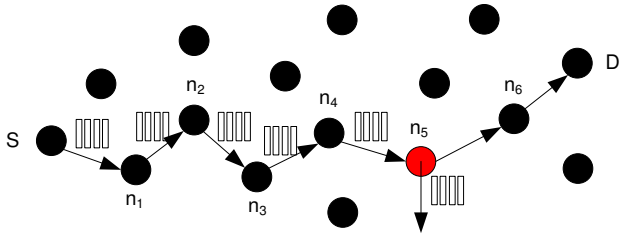


Figure 2.1: The source sends traffic to D along P_{SD}. Node n_5 drops all packets.

address the problem of identifying nodes in M and providing publicly verifiable proof of misbehavior. Our goal is to achieve misbehavior identification in a resource-efficient manner. In Figure 2.1, the source sends packets to the destination through a path $P_{SD} = \{n_1, \ldots, n_6\}$. Node n_5 misbehaves by dropping all packets. We are interested in identifying n_5 and constructing proof of its misbehavior.

CHAPTER 3

Identification of a Single Misbehaving Node

3.1 Introduction

In this chapter, we present a misbehavior identification scheme that locates a single misbehaving node along a problematic path P_{SD}. Let a source S send packets to a destination D along a path P_{SD}, with node $n_M \in P_{SD}$ dropping all packets.

The source audits several nodes in P_{SD} when notified of a performance drop. The purpose of these audits is to have nodes commit to the set of packets they forwarded to the next hop. The source collects multiple audit replies, combining them in order to identify the link on which packet forwarding fails. This is accomplished by progressively limiting a set of suspicious nodes, in which nodes upstream from the misbehaving nodes report forwarding all packets while nodes downstream from the misbehaving node report forwarding no packets. When audited, transmission of the entire set of forwarded packets back to the source requires high communication and storage requirements. Thus we utilize Bloom filters, which are compact membership sets and hence provide a storage-efficient way of performing membership testing for the set of packets forwarded to the next hop.

The identification scheme consists of three processes: (a) the search process, (b) the audit process, and (c) the identification process. We first describe the search process which is responsible for selecting nodes to be audited. We then describe the audit process and how audited nodes commit to publicly verifiable proofs of the set of packets forwarded to the next hope. Lastly, we show how the source identifies the misbehaving node, resulting in the publicly verifiable proof of misbehavior.

3.2 The Search Process

The goal of the search process is to select nodes to be audited such that the source converges on two neighboring nodes, one that claims the misbehaving node is upstream and one that claims the misbehaving node is downstream. Let $|M| = 1$ with the misbehaving node being defined as n_M. 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 $\mathcal{V}$. Let the source audit nodes by asking if they have received and forwarded the set of packets X_S sent by the source for a given duration. When the source audits node $n_i \in P_{SD}$, either: (a) n_i claims to have received and forwarded all packets in X_S, or (b) n_i claims to have not received packets in X_S from the previous hop. For each case, the following conclusions can be drawn.

If n_i claims to have received and forwarded all packets in X_S, the source concludes 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. Therefore, the source reduces the suspicious set $\mathcal{V}$ to $\{n_i, \ldots, n_k\}$. 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 towards the destination.

If n_i claims to have not received packets in X_S from the previous hop, the source concludes that all downstream nodes are honest. This is true, since if any downstream node n_j is misbehaving, all nodes upstream from n_j, including n_i, would claim to have received and forwarded all packets in X_S, given that only one misbehaving node exists in P_{SD}. Since n_i claims not to have received packets in X_S, the source concludes that the misbehaving node n_M must be upstream of n_i. Thus, the source reduces the suspicious set $\mathcal{V}$ to $\{n_1, \ldots, n_i\}$.

By repeated audits, the source reduces $\mathcal{V}$ to two neighboring nodes, node n_i that claims the misbehaving node is downstream, and node n_{i+1} that claims the misbehaving node is upstream. Each audited node n_i is selected such that $i = \lceil \frac{|\mathcal{V}|}{2} \rceil$.

Thus the source converges on said link after $\lceil \log_2 P_{SD} \rceil$ audits, do to the binary search. If n_M changes its behavior and forwards packets honestly, the source will be alerted by the destination to this change. Hence the source will pause the search until it is alerted by the destination that misbehavior is occurring again.

3.3 The Audit Process

The goal of auditing a node $n_i \in P_{SD}$ is to verify that a set of packets X_S, sent by the source S, were correctly received and forwarded by nodes upstream of n_i. Once audited, node n_i commits to the set of packets X_i that it received and forwarded to the next hop. Conflicting commitments collected from multiple nodes in P_{SD} are used to create proof of node misbehavior.

To respond to an audit, the node n_i records the packets forwarded during a given period of time, and reports them to the source. Based on this report, the source compares the packets in X_i with the packets in X_S originally sent to the destination. Buffering the packets themselves requires a large amount of storage and significant overhead for transmission back to the source. To alleviate these requirements, we employ Bloom filters which provide a storage-efficient way of performing membership testing [5]. The audit process occurs in three steps: (a) sending an audit request, (b) constructing a behavioral proof, and (c) processing the behavioral proof. We now describe these steps in detail.

3.3.1 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 auditing algorithm used. The source selects an audit duration a_d, measured in the number of packets to be audited. The value of a_d is user-definable and must be sufficiently large to differentiate misbehavior from normal packet loss rate. The source also selects an initial packet sequence number a_s, indicating the

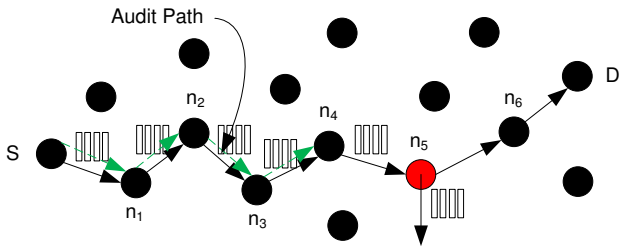


Figure 3.1: The source sends an audit request to n_i via path P_{SD}.

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 3.1, the source selects node n_4 for audit. The audit request is routed to n_4 via P_{SD}.

Note that an audit request may fail to reach the audited node n_i since a misbehaving node along P_{Sn_i} may drop it, or n_i is the misbehaving node and chooses not to respond. In this case, the source tries a threshold number of times to audit n_i. Failure to obtain an audit reply is interpreted as "Node n_i did not forward packets in X_S to the next hop." This is true since either n_i is the misbehaving node or the misbehaving node is upstream of n_i (hence the audit request and other packets were dropped).

3.3.2 Constructing a Behavioral Proof

When a node is audited, it constructs a behavioral proof of the set of all packets it forwards, from a_s to $a_s + a_d$, denoted by $X = \{x_{a_s}, x_{a_s+1}, \ldots, x_{a_s+a_d}\}$. Buffering packets themselves would require a large amount of storage and significant overhead for transmission back to the source. On the other hand, Bloom filters [5] provide a compact representation of membership for a set $X = \{x_{a_s}, x_{a_s+1}, \ldots, x_{a_s+a_d}\}$ 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 z independent hash functions $h_l, 1 \leq l \leq z$ with each h_l having a range of $\{1, \ldots, m\}$. The corresponding bits $h_l(x_i), 1 \leq l \leq z$ of vector v are set to one. To check if y is

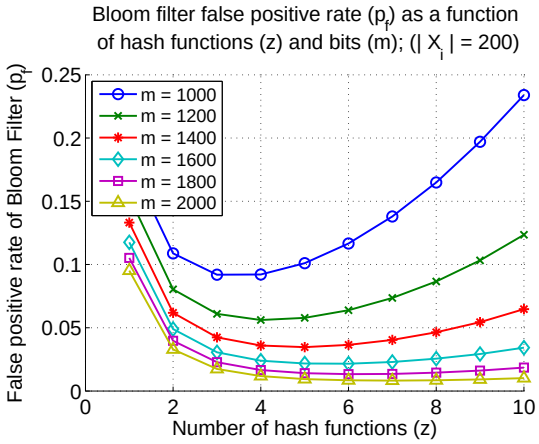


Figure 3.2: False positive rate of a Bloom filter as a function of the number of hash functions z and the length of the Bloom filter in bits m.

a member of X, element y is hashed z 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 [6]:

$$p_f = \left(1 - \left(1 - \frac{1}{m}\right)^{zN}\right)^z \approx \left(1 - e^{-\frac{zN}{m}}\right)^z \tag{3.1}$$

In Figure 3.2, we show the false positive rate p_f as a function of the number of hash functions z and the length of the Bloom filter in bits m. In Figure 3.3(a), we show a Bloom filter v ($m = 10$) that has been initialized to zero, representing the membership set $X = \{\}$. Figure 3.3(b), shows element x_1 being added to v by passing through z independent hash functions $h_l, 1 \leq l \leq z$, 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 3.3(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

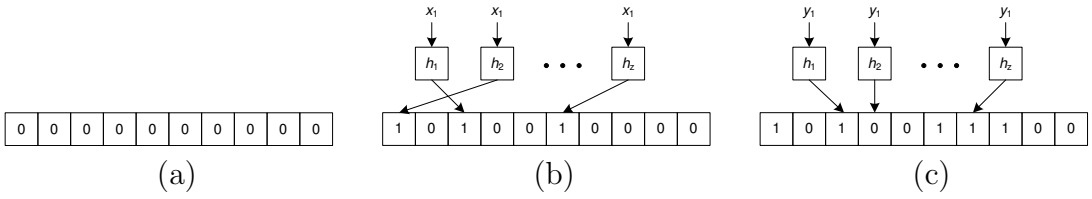


Figure 3.3: (a) Initialize Bloom Filter (b) x_1 is added to Bloom filter by passing through z hash functions with corresponding bits set to zero (c) Since $h_2(y_1)$ corresponds to a zero bit, y_1 not in Bloom filter.

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 $x_j, a_s \leq j \leq (a_s + a_d)$ into its Bloom filter v_i. After a_d packets have been added to v_i, n_i signs v_i and sends it to the source via P_{SD}. 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 $x_j,\ a_s \leq j \leq (a_s + a_d)$ are added to v_S. When the source 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_S, and v_i from node n_i.

3.3.3 Processing the Behavioral Proof

When the source 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, the source may re-transmit the request using alternative paths. After a certain number of reply failures, the source 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_S, v_i by computing the inner

product $\langle v_S, v_i \rangle$, which measures the similarity between vectors v_S, v_i. 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 [6]:

$$\begin{aligned} \langle v_S, v_i \rangle \;\; \approx \;\; m \Bigg(1 - \left(1 - \frac{1}{m}\right)^{z|X_S|} - \left(1 - \frac{1}{m}\right)^{z|X_i|} \\ + \left(1 - \frac{1}{m}\right)^{z(|X_S|+|X_i|-|X_S \bigcap X_i|)} \Bigg). \end{aligned} \tag{3.2}$$

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

$$\begin{aligned} |X_S \bigcap X_i| \;\; \approx \;\; |X_S| + |X_i| - \frac{\log\left(\frac{<v_S, v_i>}{m}\right.} \\ \frac{\left. + \left(1 - \frac{1}{m}\right)^{z|X_S|} + \left(1 - \frac{1}{m}\right)^{z|X_i|}\right)}{z \log\left(1 - \frac{1}{m}\right)}. \end{aligned} \tag{3.3}$$

The cardinality of $X_S \bigcap X_i S$ provides a method to verify if packets in X_S are in X_i. Furthermore, the source 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, $|X_S \bigcap X_i| \approx |X_S|$ since $X_S \subseteq X_i$ and any membership test would come out positive. However, the source can easily verify if Bloom filter X_i contains packets not in X_S. 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, the source 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-$(p_f)^r$.

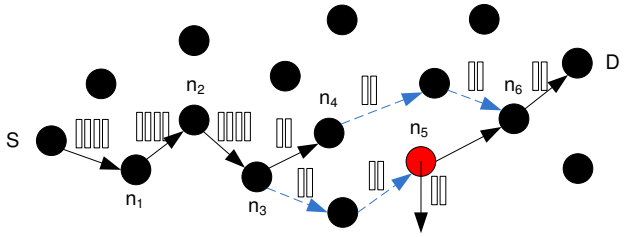


Figure 3.4: The search converges on link (n_4, n_5). S makes a slight alteration to P_{SD}, isolating n_4 and n_5, to determine that n_5 is the misbehaving node.

3.4 The Identification Process

Once the search process has converged on the link (n_i, n_{i+1}) in which node n_i that claims the misbehaving node is downstream, and node n_{i+1} that claims the misbehaving node is upstream, the two nodes n_i, n_{i+1} are excluded in turn from the routing path to the destination. The node n_{i-1} will split the traffic between n_i, n_{i+1} in turn. In Figure 3.4, S uses node n_3 to exclude in turn nodes n_4 and n_5. The source alerts the destination that two nodes are monitored. The destination reports to the source which of the two paths misbehavior is occurring on and alerts the source. The source uses this alert from the destination to identify the misbehaving node.

3.5 Collusion of Multiple Misbehaving Nodes

We have described how a source can identify a misbehaving node in P_{SD}. However, it is possible for an honest node to be framed of misbehavior if P_{SD} contains two colluding misbehaving nodes. For example, let assume the source is executing the misbehavior identification scheme and there exists collusion between misbehaving nodes. In Figure 3.5(a), nodes n_1 and n_3 are misbehaving and colluding. Initially, n_1 drops packets while n_3 behaves honestly. The source audits n_3, who replies that an upstream node is misbehaving. In Figure 3.5(b), the source selects n_2 for audit. Nodes n_1, n_3 collude, altering their behaviors such that n_3 drops all packets while

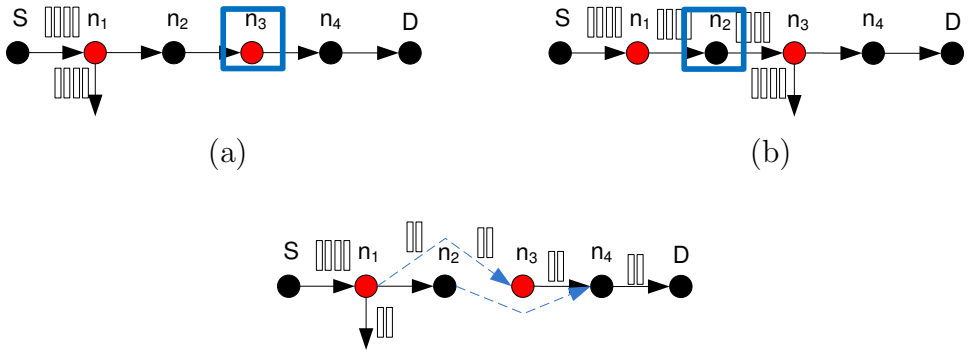


Figure 3.5: (a) Nodes n_1, n_3 are colluding nodes, with n_1 dropping all packets. Node n_3 is audited, claiming misbehavior is upstream. (b) Nodes n_1, n_3 collude to alter their behavioral strategy, i.e., n_1 behaves honestly while n_3 drops all packets. The audited node n_2 claims misbehavior is downstream. (c) The nodes n_2, n_3 are excluded in turn during packet forwarding. Since n_1 is misbehaving, it drops n_2's packets, causing n_2 to be accused of misbehavior.

n_1 behaves honestly. Thus, the audited node n_2 reports misbehavior is occurring downstream and the source has limited the possible set of misbehaving nodes to $\{n_2, n_3\}$. In Figure 3.5(c), the source makes a slight modification to the routing path, such that packets are routed through n_2 and n_3 in turn. Node n_1 can now selectively drop all packets routed through n_2, thus making n_2 appear as the misbehaving node.

This problem arises since misbehaving nodes n_1 and n_3 can share information about the state of the search and modify their behavioral strategies to accuse the honest node n_2 of misbehavior. Even if the source randomizes the set of packets X_S to prevent a node from conjecturing upon its value, this does not prevent the sharing of information when misbehaving nodes are audited. Thus, n_1, n_3 can *lie* to the source by changing where the misbehavior appears in P_{SD}.

CHAPTER 4

REAct: A Resource-Efficient Accountability Scheme for Misbehavior Identification based on Rényi-Ulam Games

4.1 Introduction

In the previous chapter, we presented a misbehavior identification scheme in which the source audits nodes along the routing path, forcing them to commit to the set of packets they forward to the next hop. The source collects multiple audit replies to identify the misbehaving node. We noted that although an honest node will always respond faithfully to an audit request, a misbehaving node can lie about the set of packets forwarded. Provided sufficient replies from honest nodes, the source can identify the misbehaving ones. The process of analyzing audit plies and selecting the nodes to be auditing is analogous to Rényi-Ulam games [53, 60, 67].

Rényi-Ulam games are searching games between two entities, a questioner and a responder. The responder selects a secret value from a finite search space. The questioner attempts to determine the responders secret value by asking questions, in which the responder can lie up to a pre-determined number of times. The questioner wins the game if it can determine the responders secret value after at most q questions.

In this chapter, we map the misbehavior identification problem to Rényi-Ulam games. Solutions to Rényi-Ulam games are designed to use the least number of questions possible. By basing our auditing strategies on these games, we are able to reduce the number of audits required to identify misbehaving nodes. Moreover, we show that such auditing strategies are collusion-resistant.

Based on Rényi-Ulam games, we develop REAct, a resource-efficient account-

ability scheme for node misbehavior. We describe how the source utilizes the corresponding questioning strategies to identify misbehaving nodes in P_{SD}. We formulate each case, starting from a single misbehaving node and proceed to the cause of multiple colluding ones. Audits are performed as described in Section 3.3. We first describe relevant background on Rényi-Ulam games.

4.2 Rényi-Ulam Games

Rényi-Ulam games are searching games independently proposed by Rényi [60] and Ulam [67]. These games involve two players; a questioner and a responder. The responder selects a secret value ω from a finite search space Ω. The questioner attempts to determine ω by asking at most q questions to which the responder is allowed up to ℓ lies.

Before starting the game, the players agree on: (a) the search space Ω, (b) the number of questions q allowed to the questioner, (c) the number of lies ℓ allowed to the responder, and (d) the nature of the interaction between the players. The latter is defined by the type of questions allowed and the way they are asked. The format of the questions can be classified into three categories: (a) bit questions, (b) cut questions, and (c) membership questions. Bit questions are defined as "Is the ith-bit of ω equal to 1?" Cut questions are defined as, for some $y \in \Omega$, "Is $\omega \leq y$?" Membership questions are defined as, for some subset $A \subseteq \Omega$, "Is $\omega \in A$?" The same questioning format is assumed for the entire duration of the game.

Two modes are possible for the interaction between the two players; a batch mode and an adaptive mode. In batch mode, the questioner must submit 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.

In some versions of the game, restrictions are placed on the responder as to when and how it may use its lies. As an example, the total number of lies after i questions

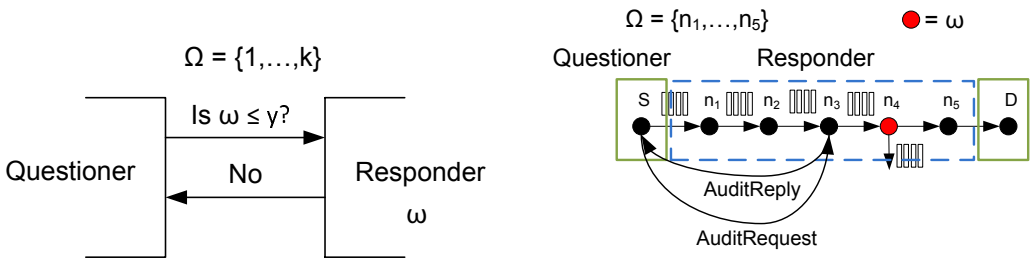


Figure 4.1: (a) A generic Rényi-Ulam game. (b) Misbehavior identification mapped to a Rényi-Ulam game.

must be less than some fraction of i. The questioner wins the game if it determines ω after at most q questions. Else, the responder wins. The questioner is said to have a "winning strategy" if it can always win the game after at most q questions, regardless of how the responder lies.

4.2.1 Applications of Rényi Ulam Games

A major area of application for Rényi-Ulam games is in error correcting codes [4, 51, 53]. Assume two nodes are transmitting messages through a noisy channel. The sender sends a message through noisy channel, which can invert bits causing the receiver to receive a corrupted message. In this scenario, the responder is represented as the noisy channel, and lies by inverting up to a fixed number of bits. The questioner, represented by the receiver, attempts to recover the original messages. Obviously, if the receiver wants the correct a corrupted message with at most e errors, then it requires the use of an e-error correcting code. These e-error correcting codes can be modeled after Rényi-Ulam searching games. This simply idea can be further expanded to a scenario in which the sender receives feedback from the receiver concerning the transmission of messages. This feedback is also transmitted through the noise channel. Thus the sender must decided on the optimal transmission rate in which sent messages are represented as questions and the

(possibly corrupted) feedback represents answers.

Additionally, Rényi-Ulam games can be applied to communication networks in which some of the components are faulty [52, 66]. Two common types of faults are *crash* and *Byzantine* faults. A crash fault is relatively harmless in the sense that it does not send, receive, or alter any messages in the network. Byzantine faults, by contrast, are components which can either stop, reroute, or alter messages arbitrarily (or maliciously). One such scenario is a distributed computing network in which each machine has an initial value. All fault-free machines must agree on the same value in a logical way. Thus faulty machines may attempt to prevent such a convergence on a value. Another example is a collection of nodes in a network, each of which have a message. The goal is to propagate the information such that when finished, each node has a copy of all the other nodes messages. A faulty node may attempt to prevent message transfers or transmit altered messages.

Rényi-Ulam games are also used to address the problem of searching in graphs containing uncertainty [23, 33, 34, 36]. Assume an undirected graph in which one of the nodes contains a sought after token (or piece of information). The questioner starts at some node and attempts to located the token by traveling along as few edges as possible. Each node that does not contain the token offers advice to the questioner about which of its edges is on the shortest path to the token. However, this advices is unreliable. Analogies to this problem would be asking for directions at each stop in which the responder may lie (or give incorrect advice) with a fixed probability. This problem can also be thought of as a user searching the Internet looking for some piece of information. In this scenario, web pages may contain incorrect or misleading links, while the user attempts to find the token in the least number of "clicks" as possible.

4.3 Mapping to Rényi-Ulam Games

In our mapping of misbehavior identification to Rényi-Ulam games, the role of the questioner is assumed by the source and the destination, while the role of the responder is assumed by the nodes in P_{SD} (by P_{SD}, we refer to all intermediate nodes excluding S and D). The search space is defined as the set of nodes in P_{SD}, i.e., $\Omega = \{n_1, \ldots, n_k\}$ with $k = |P_{SD}|$. The responder selects a number $\omega \in \{1, \ldots, k\}$, corresponding to the node n_ω in path P_{SD} which is misbehaving. The source's goal is to determine the value of 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 bit, cut, or membership questions. The responder lies when a misbehaving node lies with respect to the packets forwarded to the next hop. For example, a node lies by either claiming to forward all packets received when in reality it drops them, or claiming to have forwarded no packets indicating that no packets were received from the previous hop. The location of the misbehaving nodes in P_{SD} is mapped to the placement of such lies by the responder. Figures 4.1(a) and 4.1(b) illustrate the mapping of the misbehavior identification problem to a Rényi-Ulam game.

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

4.4 Rényi-Ulam Inspired Searching Strategies

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

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

Proposition 1. *At least one of the incident nodes to a misbehaving link is misbehaving.*

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

Definition 2. *A simultaneous audit is defined as the process of auditing two or more nodes with respect to the same set of packets X_S.*

Corollary 1. *Two behaving nodes cannot be incident to a misbehaving link when simultaneously audited.*

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

Note however that the converse statement to Corollary 1 is not true. For two nodes n_i and n_{i+1} in which $|X_i \bigcap X_{i+1}| \approx |X_i|$, we cannot conclude that both nodes are honest. This is due to the fact that two colluding misbehaving nodes may be incident to a link, thus claiming similar audit replies regardless of the packets forwarded downstream. We use the notions of a misbehaving link and simultaneous audit to identify the misbehaving nodes. We now explore efficient strategies for identifying the misbehaving node(s), inspired by Rényi-Ulam games.

4.4.1 REAct-B: Batch Audits

In the batch questioning game, the questioner submits all of its questions to the responder at one time. The responder can review all questions before supplying answers, allowing collusion among all questions. Each question is represented as an audit in the form, "Did node n_i receive and forward all packets in X_S honestly?" The source attempts to gather enough information about P_{SD} through the audits that at least one misbehaving link can be identified.

Let R be defined as a vector of length $|P_{SD}|$, in which $R[i] = 1$ if node n_i receives and forwards all packets in X_S honestly, and $R[i] = 0$ otherwise. Vector R is representative of the state of P_{SD}. Proposition 2 states that given R, the source can identify at least one misbehaving link.

Proposition 2. *Given R, at least one misbehaving link can be identified.*

Proof. By definition, a misbehaving link is identified by two adjacent nodes returning contradictory claims, thus $R[i] \neq R[i+1]$. For all nodes n_i upstream of n_M, $R[i] = 1$. For all nodes n_j downstream of n_M, $R[j] = 0$. Thus, regardless of the value of $R[M]$, either $(R[M-1], R[M])$ or $(R[M], R[M+1])$ will represent a misbehaving link. □

Vector R is constructed by the source through the questioning strategy. Ideally, the source would like to utilize a winning questioning strategy requiring the minimal

number of audits. Therefore, how the q questions are selected determines the amount of information about R gained by the source. As an example, if the source audits all odd numbered nodes, i.e., $n_1, n_3, \ldots$ ($q = \frac{|P_{SD}|}{2}$), the source can never win the game, since to identify a misbehaving link, its incident nodes need to be audited, which never occurs.

Thus, the source needs to guarantee that the q questions will completely construct R, such that by Proposition 2, the game will be won and the misbehaving link identified. A single audit reply will determine one bit of R, while inferring on additional bits, i.e., if n_i claims no misbehavior this infers that all nodes upstream of n_i report no misbehavior. However, since a node may lie during its audit reply, any inference about the behavior of its upstream and downstream nodes may be incorrect. Therefore, in order to completely construct R, $q = |P_{SD}|$ audits are required. Proposition 3 shows that $q = |P_{SD}|$ represents the winning questioning strategy. The batch auditing strategy is presented in Algorithm 1.

Proposition 3. *A winning strategy exists if* $q = |P_{SD}|$.

Proof. Assume $q < |P_{SD}|$ and $|M| = 1$. There exists some node $n_i \in P_{SD}$ which is not audited. Let $R = \{1, \ldots, 1, \bullet, 0, \ldots, 0\}$, with $R[i] = \bullet$ representing an un-audited node and therefore no information. In this case, $(R[i-1], R[i])$ and $(R[i], R[i+1])$ both represent a possible misbehaving link. However, the source cannot produce proof of misbehavior since $R[i]$ is not available. Therefore the responder has won the game.

When $q = |P_{SD}|$, this results in the complete construction of R by the source, as the audit of node n_i determines $R[i] = \{1, 0\}$. By Proposition 2, given R, the source can identify at least one misbehaving node. Hence $q = |P_{SD}|$ is a winning questioning strategy. □

In Figure 4.2(a), the source sends packets to the destination along $P_{SD} = \{n_1, \ldots, n_5\}$. Node n_4 misbehaves by dropping all packets. The source audits all

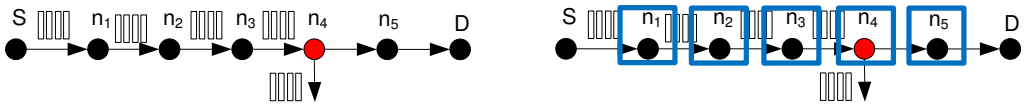


Figure 4.2: (a) S sends packets to D along P_{SD}. Misbehaving node n_4 drops all packets. (b) S audits all nodes in P_{SD}, i.e., $\{n_1, \ldots, n_5\}$ to identify the misbehaving link.

nodes in P_{SD}. If n_4 claims misbehavior is downstream ($|X_S \bigcap X_4| \approx |X_S|$), then the link (n_4, n_5) will be identified as the misbehaving link since the honest node n_5 will claim misbehavior is occurring upstream ($|X_S \bigcap X_4| \ll |X_S|$). Likewise, if n_4 claims misbehavior is upstream ($|X_S \bigcap X_4| \ll |X_S|$), then the link (n_3, n_4) will be identified as the misbehaving link since the honest node n_3 will claim misbehavior is occurring downstream ($|X_S \bigcap X_3| \approx |X_S|$). Hence the misbehaving link is identified independent of n_4's audit response.

While REAct-B provides a winning questioning strategy for the source, it requires that all nodes in P_{SD} must be audited. This has two consequences. First, misbehaving nodes are always alerted to the search, and can thus modify their misbehavior strategy to avoid implication. Secondly, message overhead increases linearly as a function of path length, since all nodes must be audited. As our metric of interest is resource-efficiency, this motivates the need for a non-linear searching algorithm with respect to message overhead. Adaptive games provide such an opportunity.

4.4.2 REAct-C: 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 in P_{SD}, i.e., $|M| = 1$ and define a suspicious set $\mathcal{V} = \{n_1, \ldots, n_k\}$.

Algorithm 1 REAct-B: Batch Questioning Algorithm

1: $Audit(n_1, n_2, \ldots, n_k)$
2: **for** $i = 0$ to $|P_{SD}|$ **do**
3: **if** $|X_i \bigcap X_{i+1}| \ll |X_i|$ **then**
4: **return** X_i, X_{i+1}
5: **end if**
6: **end for**
7: **return** *No Misbehavior*

If an audited node n_i replies with X_i such that $|X_S \bigcap X_i| \ll |X_S|$, the source concludes that all nodes downstream of n_i are behaving honestly, and therefore $n_M \leq n_i$. This is true since either n_i is honest in which case it never received packets in X_S indicating an upstream misbehaving node, or n_i is the misbehaving node lying about its audit reply. Hence, all nodes downstream of n_i must be honest and $\mathcal{V}$ is reduced to $\{n_1, \ldots, n_i\}$.

If the audited node n_i replies that $|X_S \bigcap X_i| \approx |X_S|$, the source concludes that all nodes upstream of n_i are honest, and therefore $n_M \geq n_i$. This is true, since if any node upstream of n_i was the misbehaving one, n_i would not have received packets in X_S. Thus the set $\mathcal{V}$ is reduced to $\{n_i, \ldots, n_k\}$. We note that the audited node always remains in $\mathcal{V}$, since it can lie about the forwarded packets.

Pelc [51] proposed a questioning strategy for adaptive games in which the questioner wins if either the responder's secret value ω is determined, or the questioner can prove a lie took place. For a search space of size $|\Omega|$, and a maximum number of lies ℓ, the winning strategy requires $\lceil \log_2 |\Omega| \rceil + \ell$ questions. To find ω, the questioner first performs a binary search requiring $\lceil \log_2 |\Omega| \rceil$ questions to converge to a value ω'. It then asks the responder ℓ times if $\omega \leq \omega'$. Since the responder is limited in lies, the questioner can determine if ω' is the secret value or the responder has lied.

Following the winning strategy proposed by Pelc, let the source win if either a misbehaving link is identified or the source can prove a lie has occurred. The source can converge to a single link by performing a binary search. The source initializes $\mathcal{V} = \{n_1, \ldots, n_k\}$ and selects a node $n_i \in \mathcal{V}$ to audit with $i = \lceil \frac{|\mathcal{V}|}{2} \rceil$. As previously described, $\mathcal{V}$ is reduced to either $\{n_1, \ldots, n_k\}$ or $\{n_i, \ldots, n_k\}$. The source continues to audit nodes in $\mathcal{V}$ until $|\mathcal{V}| = 2$. By Proposition 4, the source identifies a misbehaving link.

Proposition 4. *When $|M| = 1$ and n_M is continuously misbehaving, the source always converges to the misbehaving link in $\log_2(|P_{SD}|)$ audits.*

Proof. Let $|M| = 1$ with n_M being the misbehaving node. Initially, $\mathcal{V} = P_{SD}$ and hence $n_M \in \mathcal{V}$. Let the source select a node n_i upstream of n_M for audit. Since $|M| = 1$, n_i responds honestly that it forwarded packets to the next hop, reducing $\mathcal{V}$ to $\{n_i, \ldots, n_k\}$, with $n_M \in \mathcal{V}$. Similarly, if a node n_j downstream of n_M is audited, it will respond that no packets were forwarded, reducing $\mathcal{V}$ to $\{n_1, \ldots, n_j\}$. If n_M is audited, its response will indicate that misbehavior occurs either upstream of downstream. In either case $n_M \in \mathcal{V}$, since the audited node always remains in $\mathcal{V}$. The convergence of the binary search will end in a suspicious set $\mathcal{V} = \{n_{M-1}, n_M\}$ or $\mathcal{V} = \{n_M, n_{M+1}\}$, depending on whether n_M indicated that misbehavior occurs upstream of downstream. If n_M lied in its audit reply indicating that misbehavior occurs downstream of n_M (i.e. $|X_S \bigcap X_M| \approx |X_S|$), the set of suspicious nodes will be reduced to $\mathcal{V} = \{n_M, \ldots, n_k\}$. Every node in $\mathcal{V}$ besides n_M is honest and hence, when audited, will indicate that misbehavior occurs upstream. The binary search will converge on (n_M, n_{M+1}). Likewise, if n_M claims it did not forward any packets to the next hop (i.e. $|X_S \bigcap X_M| \ll |X_S|$), implying that misbehavior occurs upstream of n_M, the binary search will converge on link (n_{M-1}, n_M). In any case, the identified link is a misbehaving one since per the definition its two incident nodes provide conflicting audit replies. Since the binary search converges in $\log_2(|P_{SD}|)$, in case $|M| = 1$ the source will locate n_M in $\log_2(|P_{SD}|)$ steps. □

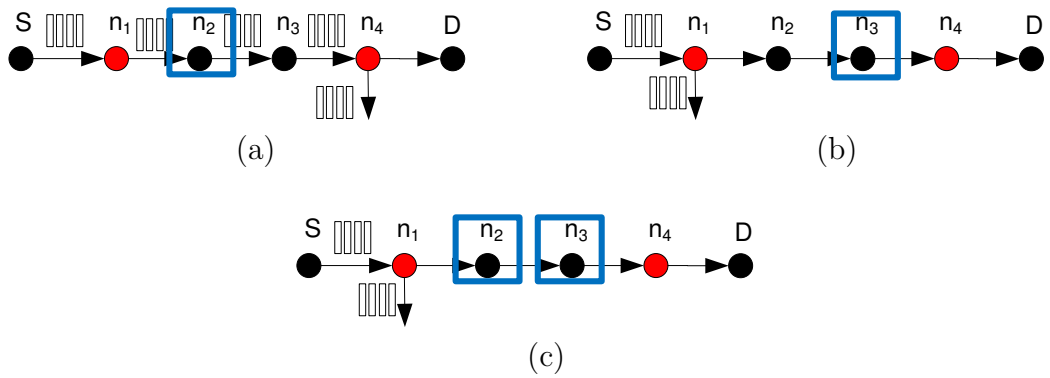


Figure 4.3: (a) Nodes n_1 and n_4 collude, with n_4 dropping all packets. Audited node n_2 claims misbehavior is downstream. (b) Nodes n_1 and n_4 alter their behaviors, with n_1 dropping all packets and n_4 behaving honestly. Audited node n_3 claims misbehavior is upstream. (c) Source simultaneously audits n_2 and n_3 to verify if misbehaving link exists.

During the execution of REAct-C, the misbehaving node(s) may change their misbehavior strategy by forwarding all packets (behaving honestly) to the next hop honestly. In such a case, the destination alerts the source of the absence of misbehavior in P_{SD} via an alarm message. The source will take two steps, independent of the auditing strategy used. First, any outstanding audits will be discarded. Second, REAct-C will be suspended at it's current state until the source receives an alarm message from the destination notifying it that misbehavior is occurring again. When misbehavior is resumed, the source randomly selects a new set X_S of packets for the next audit. Thus a misbehaving node cannot conjecture when the audit process begins, preventing it from modifying its dropping pattern accordingly.

However, Proposition 4 assumes the source is aware of the number of misbehaving nodes in P_{SD}, which is not true. If two or more nodes collude, the source may converge on a link in which both nodes are behaving, as shown in the following example. In Figure 4.3(a), $M = \{n_1, n_4\}$ with nodes n_1, n_4 colluding. Initially, n_4 is dropping all packets, while n_1 behaves. Let node n_2 be audited and report no misbehavior, thus $\mathcal{V} = \{n_2, n_3, n_4\}$. Assume now that nodes n_1, n_4 switch their

behavior with node n_1 dropping packets while n_4 is behaving, as shown in Figure 4.3(b). If node n_3 is audited, it will report misbehavior upstream, reducing $\mathcal{V}$ to $\{n_2, n_3\}$ and thus removing misbehaving node n_4 from $\mathcal{V}$. Hence, link (n_2, n_3) is incorrectly identified as the misbehaving one.

Pelc solves this problem through the repetitive questioning of the result obtained via binary search, thereby exhausting the responder's lies. In our case, a simultaneous audit on nodes n_i, n_{i+1} of an identified link $\mathcal{V} = \{n_i, n_{i+1}\}$ is sufficient to identify a misbehaving link or the occurrence of a lie. If $|X_i \bigcap X_{i+1}| \ll |X_i|$, a misbehaving link is identified. If $|X_i \bigcap X_{i+1}| \approx |X_i|$, the source concludes that a lie occurred, since in the absence of lies, the binary search would have converged to the misbehaving link. Returning to our previous example, in Figure 4.3(c), n_2, n_3 are simultaneously audited. Since both nodes are honest, they return identical audit replies and no misbehaving link is identified. In this example, the responder has lied by changing the value of ω during the search, i.e., initially $\omega = n_4$, then $\omega = n_1$. However, S identifies a lie occurred since n_2 and n_3 both claim misbehavior is upstream.

When the source identifies a lie occurred, is can also reach to the following conclusion: either (a) $n_M \in \mathcal{V}$ but lied during the simultaneous audit, or (b) $|M| \geq 2$ with at least one misbehaving node upstream of n_{i+1} and one downstream of n_i. Note that if $|M| = 1$ and the misbehaving node stops misbehaving (due to the fact that it is being audited) the destination alerts the source that misbehavior has stopped in P_{SD} via an alarm message. If the destination does not alert the source that performance in P_{SD} has been restored, the source concludes that $|M| \geq 2$. This is evident in our example by the responses of n_2 that on the first audit in Figure 4.3(a), it claims that misbehavior is downstream, while in Figure 4.3(c), it claims misbehavior is upstream.

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 and one misbehaving node

is downstream of n_{i+1}, it attempts to isolate the effect of the misbehavior of each node by partitioning P_{SD} into $P_{Sn_i} = \{n_1, \ldots, n_i\}$ and $P_{n_{i+1}D} = \{n_{i+1}, \ldots, n_k\}$. The source repeats the auditing strategy recursively for each path partition $P_{Sn_i}, P_{n_{i+1}D}$. However, note that the destination can only determine if misbehavior occurs in P_{SD}; not which partition.

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

Likewise on $P_{n_{i+1}D}$, the audit response of n_{i+1} acts as a verification if packets from X_S have reached this partition. Node n_{i+1} therefore acts as a pseudo-source for $P_{n_{i+1}D}$. Much like n_i, if n_{i+1} lies it immediately implicates itself in a misbehaving link since (n_i, n_{i+1}) is always audited. Thus the source can identify multiple misbehaving links using this adaptive auditing strategy. REAct-C is presented in Algorithm 2.

4.4.3 REAct-M: Adaptive Audits with Membership Questions

The source can also use an adaptive auditing strategy based on membership questions to identify the misbehaving nodes. Membership questions can be constructed by combining two cut questions. To answer the membership question, "Is $n_M \in A = \{n_i, \ldots, n_j\}$?" the source audits n_i, n_j simultaneously and compares their audit replies. If $|X_i \bigcap X_j| \approx |X_i|$, then n_i, n_j claim that $n_M \notin A$, since all

Algorithm 2 REAct-C: Cut Questioning Algorithm

1: $n_i \leftarrow n_1, n_j \leftarrow n_{|P_{SD}|}, \mathcal{V} = \{n_i, \ldots, n_j\}$
2: **while** $|\mathcal{V}| > 2$ **do**
3: $h = \lceil \frac{|\mathcal{V}|}{2} \rceil,\ Audit(n_h)$
4: **if** $|X_S \bigcap X_h| \approx |X_S|$ **then**
5: $n_i \leftarrow n_h$
6: **else**
7: $n_j \leftarrow n_h$
8: **end if**
9: **end while**
10: $Audit(n_i, n_j)$
11: **if** $|X_i \bigcap X_j| \ll |X_i|$ **then**
12: **return** X_i, X_j
13: **else**
14: **return** $|M| \geq 2, Partition\ P_{SD}$
15: **end if**

packets forwarded by n_i are received by n_j. Else, they claim $n_M \in A$.

Dhagat et al. [17] proposed an adaptive questioning strategy which proceeds in stages. During each stage, the questioner either believes the responder's answer and places it in a trusted set T, or discards it if the answer contradicts prior answers. Let $\mathcal{V}_j$ represent the set of possible values for ω at stage j, with $\mathcal{V}_1$ being initialized to Ω.

Suppose that $\mathcal{V}_j$ is the current stage, with $|\mathcal{V}_j|$ greater than one, and let set $\{r_{j-1,a}, r_{j-1,b}\}$ represent the answers to round $j-1$. The questioner divides $\mathcal{V}_j$ into two equal-sized subsets, A and B. The responder is asked "Is $\omega \in A$?" If the answer $r_{j,a}$ is "yes", the questioner adds $\{r_{j,a}\}$ to T and moves to the next stage with $\mathcal{V}_{j+1} = A$. If the responder replies "no", the questioner asks "Is $\omega \in B$?" If the

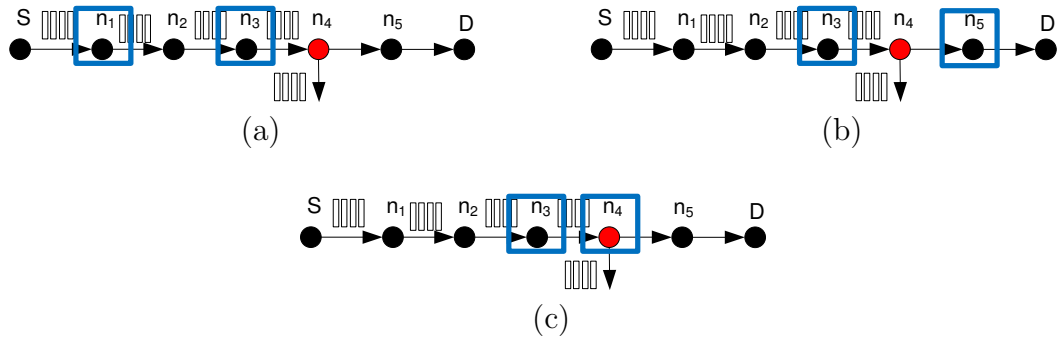


Figure 4.4: (a) Let $\mathcal{V}_1 = \{S, n_1, \ldots, n_5, D\}$ with $A = \{S, n_1, n_2, n_3\}, B = \{n_3, n_4, n_5, D\}$ and $n_M = n_4$. The source audits A, concluding $n_M \notin A$. (b) The source then audits B, concluding $n_M \in B$. (c) The source proceeds to stage $\mathcal{V}_2 = \{n_3, n_4, n_5, D\}$ and continues the auditing strategy.

answer $r_{j,b}$ is "yes," $\{r_{j,a}, r_{j,b}\}$ are added to T and the questioner moves to $\mathcal{V}_{j+1} = B$. If both $r_{j,a}, r_{j,b}$ are negative, then the questioner removes $\{r_{j-1,a}, r_{j-1,b}\}$ from T, and returns to stage $\mathcal{V}_{j-1}$. The questioner then selects a different partition of $\mathcal{V}_{j_1}$ for stage j and repeats the questioning on each partition. Dhagat et. al. showed that the responder's secret value ω can be identified after $q = \lceil \frac{2\log_2|\Omega|}{1-3\beta} \rceil$ questions, when $\beta < \frac{1}{3}$, with β being the fraction of q than are lies [17]. To prevent the continuous lies from the same misbehaving node, the source selects a new n_i' and repeats the membership questions, until $|\mathcal{V}_j| = 2$.

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

a lie has occurred.

In Figure 4.4(a), node n_4 is the misbehaving node. The source splits $\mathcal{V}_1 = \{S, n_1, \ldots, n_5, D\}$ to sets $A = \{S, n_1, n_2, n_3\}, B = \{n_3, n_4, n_5, D\}$, and audits n_3 to realize the membership question "Is $n_M \in A$?" Since $|X_S \bigcap X_3| \approx |X_S|$, the source asks "Is $n_M \in B$?" by simultaneously auditing n_3, D, as shown in Figure 4.4(b). The outcome of the new audit is $|X_3 \bigcap X_D| \ll |X_3|$, and the the source concludes $n_M \in B$. In Figure 4.4(c), the source moves to the next auditing stage, with $\mathcal{V}_2 = B = \{n_3, n_4, n_5, D\}$, by dividing $\mathcal{V}_2$ into two memberships sets. The process is repeated until $|\mathcal{V}_j| = 2$. In our example, the source converges on the misbehaving link $\{n_3, n_4\}$.

Proposition 5. *When $|M| = 1$, the source converges to the misbehaving link in less than $4\log_2(|P_{SD}|) + 2$ audits.*

Proof. Let the source be at stage $\mathcal{V}_j = \{n_i, \ldots, n_k\}$ with $n_M \in \mathcal{V}_j$ and select node n_h for audit, creating membership sets $A = \{n_i, \ldots, n_h\}$ and $B = \{n_h, \ldots, n_k\}$. If $n_M \neq n_i, n_h, n_k$, then all audit responses will be honest and the source will conclude either $n_M \in A$ or $n_M \in B$, thus proceeding to the next stage with $\mathcal{V}_{j+1} = A$, $\mathcal{V}_{j+1} = B$ and $n_M \in \mathcal{V}_{j+1}$. As long as the source audits honest nodes, the set of suspicious nodes V_j will be cut to half, thus converging to $|V_j| = 2$.

Now assume one of the n_i, n_h, n_k is n_M. When audited, n_M will either respond honestly, or lie. If n_M responds honestly, this is equivalent to both audited nodes being honest and the search will proceed to state $\mathcal{V}_{j+1}$ with $n_M \in \mathcal{V}_{j+1}$ and $|\mathcal{V}_{j+1}| = \frac{|\mathcal{V}_j|}{2}$. Thus the search continues to converge. If n_M lies, the source will obtain negative answers from both membership questions, unable to reduce $\mathcal{V}_j$ further, thus returning to stage $\mathcal{V}_{j-1}$ with $n_M \in \mathcal{V}_{j-1}$. The source will then pick a different n_h, and repeat the set splitting and auditing, thus preventing the same lie from repeating. Since there is only one misbehaving node, and each node is audited at most once, the auditing strategy will converge to $|\mathcal{V}_j| = 2$, with n_M in $\mathcal{V}_j$.

For computing the number of steps needed for the convergence to the misbehaving link, assume the worst case and let each stage always require two membership questions, i.e., "Is $n_M \in A$?" and "Is $n_M \in B$?" In the absence of lies, the total number of membership questions needed is $2\log_2(|P_{SD}|)$. This is true, since at each stage we split the suspicious set to half similar to a binary search. To realize a membership question we need to simultaneously audit two nodes, requiring a total of $4\log_2(|P_{SD}|)$ audits in the worst case (assume no lies). If n_M is audited and lies, the search backtracks to the previous stage, resulting in the waste of two audits. Given that $|M| = 1$ and the fact that the source always selects a different node to audit after a backtrack, the misbehaving node will be audited only once. Thus, in the worst case, the source requires $q \leq 4\log_2(|P_{SD}|) + 2$ audits. □

REAct-M is presented in Algorithm 3. Now assume $|M| \geq 2$. By Corollary 2, if the source converges to a link, it must be misbehaving.

Corollary 2. *The source can never converge to a link containing two behaving nodes.*

Proof. According to REAct-M, the source must receive conflicting reports from two simultaneously audited nodes to proceed from stage $j-1$ to stage j. Hence, for terminating to a set $\mathcal{V}_j = \{n_i, n_{i+1}\}$ the source must receive conflicting audit replies from n_i, n_{i+1}, when simultaneously audited. However, as shown in Corollary 1, this cannot occur if both n_i, n_{i+1} are behaving nodes. □

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

Algorithm 3 REAct-M: Membership Questioning Algorithm

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

4.5 Node Identification

Regardless of the auditing strategy employed, the source will identify a misbehaving link (n_i, n_{i+1}), in which either n_i or n_{i+1} is sure to be the misbehaving node. The source identifies the misbehaving node by making a slight alteration to the routing path P_{SD}. We now present two methods of path modification, (a) Path Division, and (b) Path Expansion. We then present how the source identifies which of the two nodes is causing the misbehavior for both the single and multiple misbehaving node scenario.

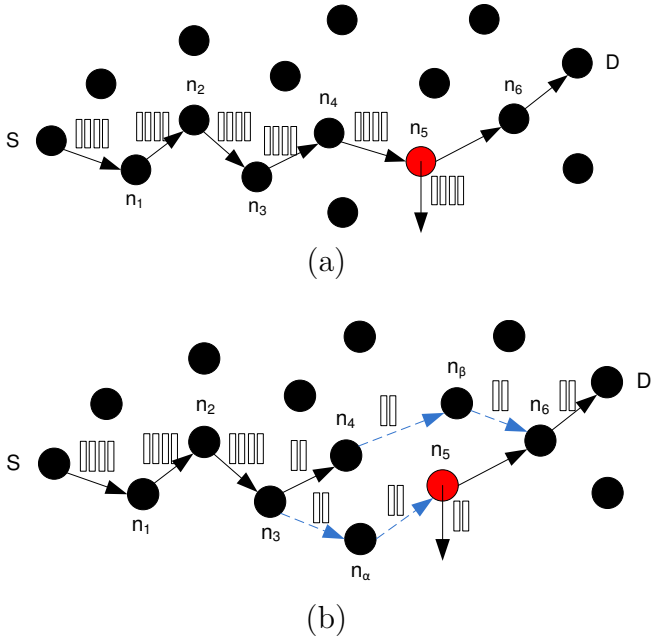


Figure 4.5: (a) Node n_5 drops packets, with link (n_4, n_5) being the misbehaving link. (b) The source performs path division separating n_4 and n_5 in independent paths, thereby isolating their affects from one another.

4.5.1 Routing Path Modification

The goal of performing routing path modification is to isolate the affect of the two nodes in the misbehaving link (n_i, n_{i+1}) such that the source can determine which node is causing the misbehavior. In the first method, called path division, the source makes a slight modification to P_{SD} such that packets are routed through either n_i or n_{i+1}. The source splits P_{SD} into two paths by selecting nodes n_j and n_k, in which $n_j, n_k \neq n_i, n_{i+1}$. Node n_j is upstream of n_i, n_{i+1}, while node n_k is downstream of n_i, n_{i+1}. The routing path segment between nodes n_j and n_k is replaced with two disjoint path segments such that one segment contains node n_i and the other segment contains node n_{i+1}. In Figure 4.5(a), the source has reduced the set of possible misbehaving nodes to two, thus either n_3 or n_4 is the misbehaving node. In Figure 4.5(b), the source selects n_2, n_5 as the upstream and

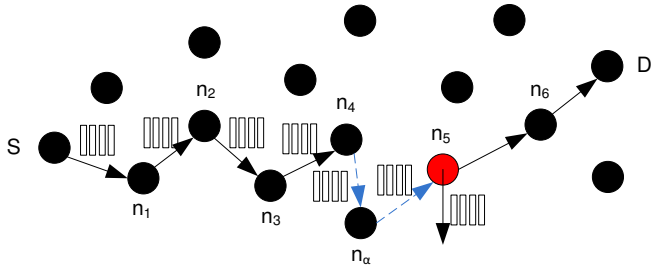


Figure 4.6: The source performs path expansion by inserting node n_α between n_4 and n_5, thereby effectively removing the misbehaving link (n_4, n_5) from P_{SD}.

downstream nodes, respectively. Path segment $\{n_2, n_3, n_4, n_5\}$ is replaced by the segments $\{n_2, n_\beta, n_4, n_5\}$ and $\{n_2, n_3, n_\alpha, n_5\}$, thus isolating n_3, n_4 from each other.

It may be the case that two independent paths are not available to perform path division. A second method, called path expansion, is then used to remove the misbehaving link (n_i, n_{i+1}) from P_{SD}. Path expansion inserts at least one additional node between the suspicious nodes n_i and n_{i+1}, thus misbehavior is forced to appear on a different link (since link (n_i, n_{i+1}) no longer exists). Again, as was shown in Figure 4.5(a), assume link (n_4, n_5) is misbehaving with node n_5 dropping packets. The source performs path expansion by adding node n_α such that $P_{SD} = \{\ldots, n_4, n_\alpha, n_5, \ldots\}$, as shown in Figure 4.6.

4.5.2 Single Misbehaving Node

Without loss of generality assume that the auditing process converged to a misbehaving link (n_M, n_{M+1}), where n_M is the misbehaving node (the other case is convergence to (n_{M-1}, n_M)). The source divides P_{SD}, into two paths such that packets are routed through either n_M or n_{M+1}, but not both, and attempts to reidentify the misbehaving link. This can be achieved by bypassing each node in P_{SD} via an alternative path. Instead of performing the entire audit process from the beginning in each of the paths, the source concentrates on the nodes around n_M, n_{M+1}, For example, in Figure 4.5(a), the source has identified link (n_3, n_4) as

the misbehaving one. In Figure 4.5(b), the source splits the traffic between two paths that bypass n_3, n_4 in turn via nodes n_α, n_β. Path segment $\{n_2, n_3, n_4, n_5\}$ is replaced by the segments $\{n_2, n_\beta, n_4, n_5\}$ and $\{n_2, n_3, n_\alpha, n_5\}$, thus isolating n_3, n_4 from each other. The source simultaneously audits nodes n_β, n_4 and n_3, n_α to identify the misbehaving link. The source identifies link (n_3, n_α) as misbehaving, and hence identifies node n_3 as the misbehaving node.

4.5.3 Multiple Misbehaving Nodes

Let us now assume the existence of multiple misbehaving nodes in P_{SD}, i.e., $|M| \geq 2$. If REAct-C is employed, the source will split P_{SD} to smaller paths in order to isolate the effect of each misbehaving node. The source can then perform the path division in each subpath as in the case of a single misbehaving node and identify where misbehavior occurs. The source uses the pseudo-sources and pseudo-destinations to perform this search. 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 REAct-M 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 process. That is, the source constructs $|\mathcal{V}_j|$ individual paths with each node in $\mathcal{V}_j$ being present on only one path.

4.6 Constructing Questions with Bloom Filters

The source combines multiple Bloom filters to realize questions, based on Equation 3.3. Assume the source asks the cut question, "Is $n_M \leq n_i$?" for some $n_i \in P_{SD}$. As previously explained, the source requests an audit from n_i, who constructs a Bloom filter v_i, representing the set of packets X_i forwarded to the next hop, and returns it to the source. The source then computes $|X_S \bigcap X_i|$. If $|X_S \bigcap X_i| \ll |X_S|$, the

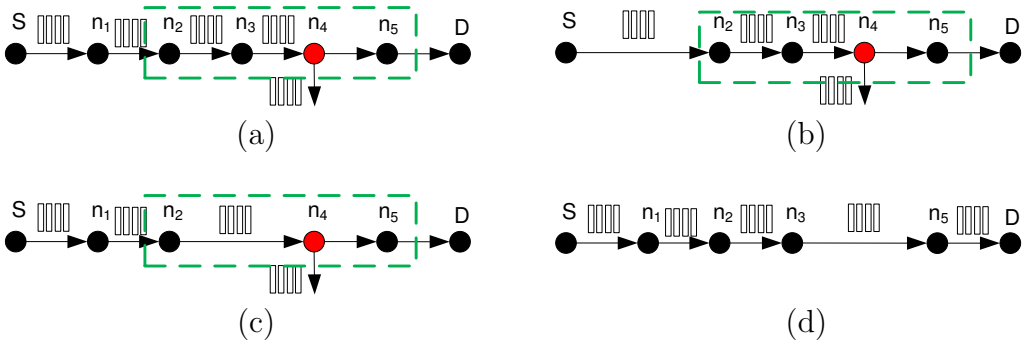


Figure 4.7: (a) S sends packets to D through P_{SD}. Node n_2 was audited, reducing the suspicious set to $\mathcal{V} = \{n_2, \ldots, n_5\}$ (shown bounded by the dotted box). Let the P_{SD} change due to mobility. (b) Honest node n_1 is removed from P_{SD}, causing no change to $\mathcal{V}$. (c) Honest node n_3 is removed from $\mathcal{V}$, causing a reduction in the suspicious set. (d) Misbehaving node n_4 is removed from $\mathcal{V}$, restoring the connection.

misbehaving node is upstream of n_i. Else, n_M is downstream of n_i.

Assume the source is using REAct-M and asks the question, "Is $n_M \in A$?" for some $A \subseteq P_{SD}$, with $A = \{n_i, \ldots, n_j\}$. As previously explained, the source requests audits from n_i and n_j, each of which constructs a Bloom filter representing the set of packets X_i and X_j forwarded to the next hop, respectively, and returns it to the source. The source then computes $|X_i \bigcap X_j|$. If $|X_i \bigcap X_j| \ll |X_i|$, then $n_M \in A$, since not all packets received by n_i were received by n_j. Thus a misbehaving node must be in A dropping packets.

4.7 Mobility

We now relax our assumption that P_{SD} does not change for the entire duration of the misbehavior identification process. Figure 4.7(a) shows a source using REAct-M, in which $\mathcal{V} = \{n_2, \ldots . n_5\}$ (shown bounded by the dotted line). Let us first look at the case where a node n_i is removed from P_{SD} during execution. If $n_i \notin \mathcal{V}$, then its removal has no effect on the auditing strategy. The source can only identify a misbehaving link from the nodes in $\mathcal{V}$. Figure 4.7(b) shows the removal of node

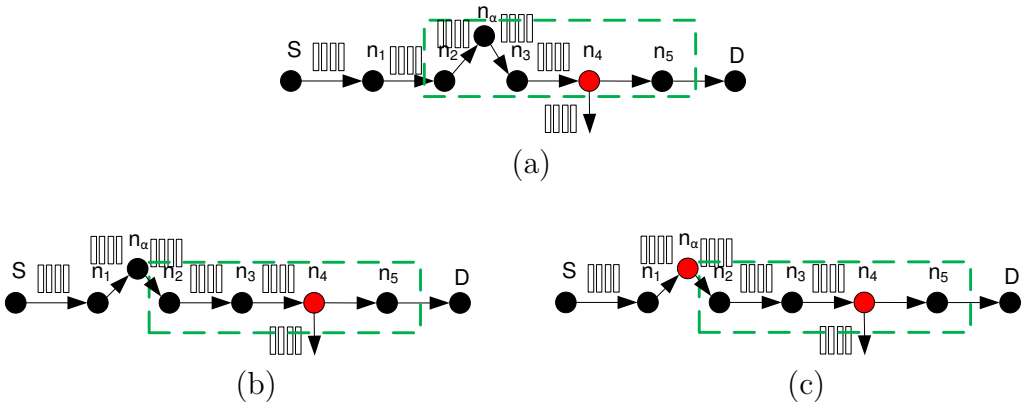


Figure 4.8: (a) Honest node n_α is added to $\mathcal{V}$; as if n_α had been there from start. (b) Honest node n_α added to P_{SD}, $n_\alpha \notin \mathcal{V}$, causing no change to $\mathcal{V}$. (c) Misbehaving node n_α added to P_{SD}, $n_\alpha \notin \mathcal{V}$. Previously shown this is detected.

$n_1 \notin \mathcal{V}$ has no affect on $\mathcal{V}$. Let $n_i \in \mathcal{V}$. There are two cases, either n_i is a behaving node, or n_i is misbehaving. If n_i is behaving, then removing n_i is analogous to reducing $\mathcal{V}$ to a smaller set that still contains the misbehaving node. Figure 4.7(c) shows the removal of honest node n_3 from $\mathcal{V}$, resulting in a beneficial reduction in the size of $\mathcal{V}$. If n_i is a misbehaving node, then the performance in P_{SD} will be restored or one less misbehaving node will be present. Figure 4.7(d) shows the removal of misbehaving node n_4 from $\mathcal{V}$ thus restoring communication between the source and destination.

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. Figure 4.8(a) shows a node n_α being added to $\mathcal{V}$, thus representing the scenario that n_α had been in $\mathcal{V}$ from the beginning of the search. Let n_i be added in P_{SD} outside $\mathcal{V}$. If n_i is an honest node, there is no effect on the audit process. Figure 4.8(b) shows an honest node n_α being added to P_{SD} outside of $\mathcal{V}$. There is no affect since n_α is honest and $\mathcal{V}$ has not changed. If n_i is a misbehaving node, then this is equivalent to the situation in which $|M| \geq 2$ and one of the n_M has been removed from $\mathcal{V}$. Figure 4.8(c) shows a misbehaving node

n_α being added to P_{SD} outside of $\mathcal{V}$. However, we have shown that both auditing strategies can recognize this scenario. In the case of cut questions, the source splits P_{SD} into two paths to isolate the misbehavior of multiple nodes. In the case of membership questions, the source will still converge on the misbehaving node that is in $\mathcal{V}$. Once this node is removed, the source will continue to identify the newly added misbehaving node.

4.8 Source/Destination Misbehavior

We now consider the case of source and/or destination misbehavior. The source and destination do not misbehave through packet dropping since if they did not want to send/receive packets, they could have refused to establish the connection in the first place. Thus we focus on source/destination misbehavior in terms of a framing attacks by producing proof of misbehavior for an honest node.

Proof of misbehavior in REAct relies on the proofs of misbehaving links. Recall that a misbehaving link is defined as two adjacent nodes that when simultaneously audited return conflicting claims of packets forwarded, i.e., link (n_i, n_{i+1}) is a misbehaving link if the result of a simultaneous audit of n_i, n_{i+1} is $|X_i \bigcap X_{i+1}| \ll |X_i|$. Thus a misbehaving source/destination would have to either forge an audit response for either n_i or n_{i+1}, since by Corollary 1, two honest nodes cannot return conflicting audit replies. However this is impossible since each audit reply is signed by the nodes private key. While the source could construct a Bloom filter X_i', it could not produce a valid signature. Thus if the source submits X_i' to nodes in the network, the signature check will fail and the claim of misbehavior will be ignored. REAct's proof of node misbehavior does not rely on nodes accusing other nodes of dropping packets, but instead only requires nodes to commit to their own behavioral in a publicly verifiable manner. Thus the proof of node misbehavior is constructed in a distributed manner and does not rely on trust in any single node.

CHAPTER 5

Performance Evaluation

5.1 Simulation Setup

We randomly deployed 100 nodes within an 80×80 square area, and randomly selected the source/destination pairs. For each source/destination pair S, D we constructed the shortest path P_{SD} and randomly selected the set of nodes that misbehave. We generated traffic from S to D according of constant bit-rate (CBR) model and sent packets to the destination following the UDP protocol. When considering scenarios in which $|M| = 2$, the misbehaving nodes behave independently of each other. Misbehaving nodes alter between behavior and misbehavior according to an ON/OFF process. The duration of each state is uniformly selected from $[1, 400]$ packets.

In our simulations, we focus on two metrics of interest:

- **Communication Overhead**: We define communication overhead as the number of messages transmitted/received by nodes in P_{SD} in order to identify the misbehaving nodes. We weigh transmitted and received messages by 1 and 0.5, respectively [20].

- **Delay**: We define delay as the time elapsed from the time instant that the source is notified of misbehavior until the misbehaving nodes are identified. We normalize delay by the audit duration.

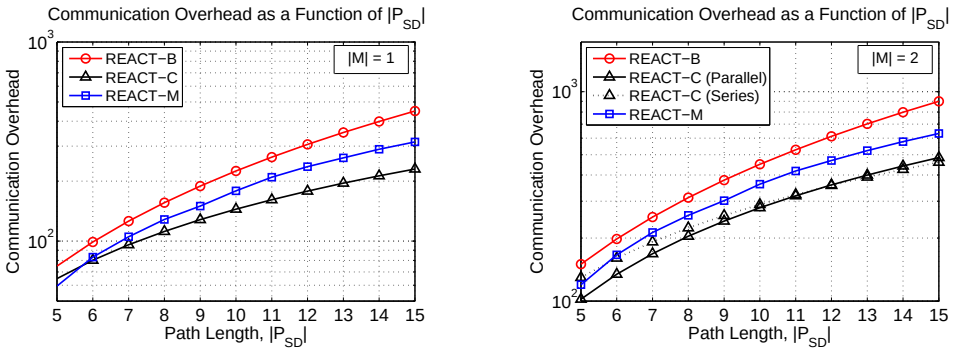


Figure 5.1: (a) Communication overhead required to identify a single misbehaving node ($|M| = 1$) as a function of path length, $|P_{SD}|$. (b) Communication overhead required to identify two misbehaving nodes ($|M| = 2$) as a function of path length.

5.2 Auditing Strategy Comparison

In our first experiment, we analyze the performance of the three Rényi-Ulam inspired auditing strategies; the strategy based on batch questions, called REAct-B, as described in Algorithm 1, the strategy based on cut questions, called REAct-C, as described in Algorithm 2, and the strategy based on membership questions, called REAct-M, as described in Algorithm 3. We compare the aforementioned strategies in terms of communication overhead and delay.

5.2.1 Communication Overhead

In Figure 5.1(a), we show the communication overhead required to identify one misbehaving node as a function of the path length. We observe that REAct-C requires less communication overhead compared to the other two. This is expected, as the realization of cut questions requires only one audit, whereas membership questions require two audits. Both auditing strategies audit in a binary fashion, thus resulting in logarithmic increases in communication overhead with respect to path length. REAct-B requires the greatest communication overhead. This is because in

batch auditing mode every node in P_{SD} must be audited to identify the misbehaving node with certainty, thus causing a linear increase of the overhead with respect to path length.

In Figure 5.1(b), we show the communication overhead required to identify two misbehaving nodes as a function of path length. Again, REAct-B requires the greatest communication overhead, due to every node in P_{SD} being audited. Depending on the misbehavior strategy of the nodes in M, REAct-C may partition P_{SD}. Thus we plot the communication overhead for two cases: (a) when no partition occurs (Series), i.e., one misbehaving node is first identified and removed before the second one can be identified, and (b) when P_{SD} is partitioned to two paths and the two paths are audited in parallel (Parallel). Note that whether REAct-C operates in series or parallel mode depends on the misbehavior patterns of the compromised nodes. Two colluding nodes can force the source to split the path, thus operating in parallel mode, while two independent misbehaving nodes may lead to a series algorithm execution. Although similar, the communication overhead of REAct-C (Parallel) increases faster than the series case since once partitioned, the source must audit two nodes at a time, i.e., the audited node n_i and the pseudo-source/destination.

5.2.2 Identification Delay

In Figure 5.2(a), we show the delay required to identify one misbehaving node as a function of path length. Both REAct-C and REAct-M incur approximately the same delay when $|M| = 1$, due to their binary search approach. REAct-B, on the other hand, requires the least delay since all nodes are audited simultaneously. Thus REAct-B requires two rounds of auditing to identify the two misbehaving links and identify the misbehaving node.

In Figure 5.2(b), we show the delay required to identify two independently misbehaving nodes as a function of path length. As expected, REAct-B incurs the least delay due to the simultaneous audit of all nodes in P_{SD}. In REAct-C, after

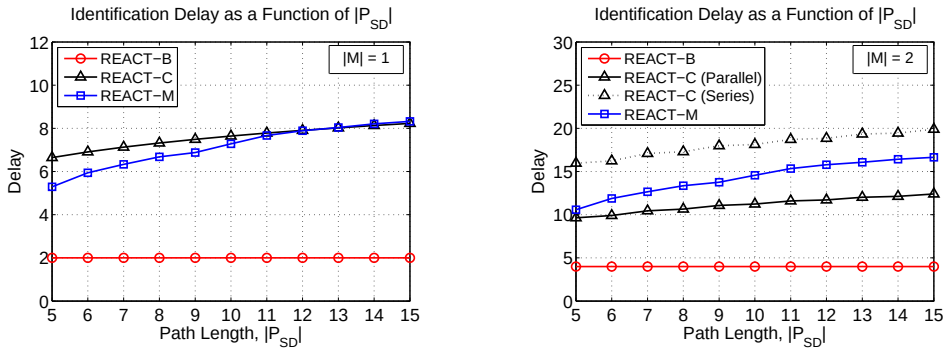


Figure 5.2: (a) Delay required to identify a single misbehaving node ($|M| = 1$) as a function of path length, $|P_{SD}|$. (b) Delay required to identify two misbehaving nodes ($|M| = 2$) as a function of path length, $|P_{SD}|$.

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

For REAct-C, we plot both the case of search in series and parallel, giving an expected range of delay. Note that the delay of REAct-M falls within this range; closer to the parallel REAct-C for smaller path sizes and closer to the series REAct-C as the path length increases. This is due to the fact that in REAct-C, the source cannot determine if a lie occurred until performing a simultaneous audit to the two nodes adjacent to the converged upon link at the end of the search. In REAct-M, 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, and not a complete series of audits in P_{SD} as in REAct-C. This results in a tradeoff in

which REAct-M incurs an additional overhead per stage compared to REAct-C by checking for contradictions at the expense of delay.

5.2.3 Impact of Node Mobility

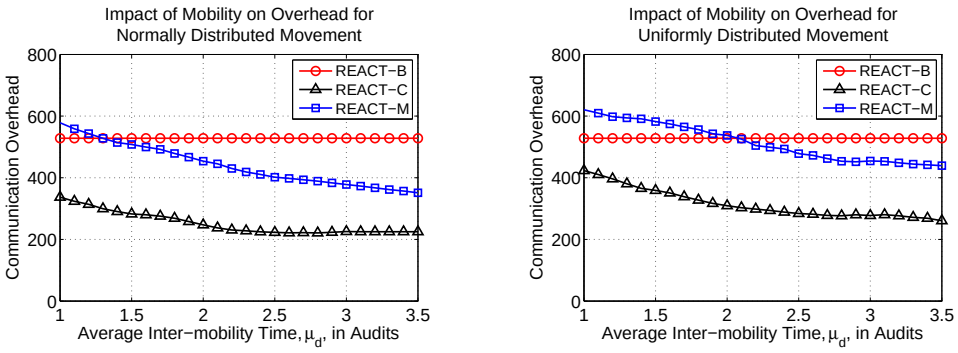
In our second experiment, we simulated the impact of node mobility on REAct in terms of communication overhead and delay. We assumed a single misbehaving node ($|M| = 1$) and focused on the case when n_M moves to a different position in the path P_{SD}. Mobility of honest nodes was not considered here since deletion of honest nodes leads to faster identification of misbehavior, and addition of honest nodes only marginally increases communication overhead and identification delay. The movement of the misbehaving node was unrestricted inside P_{SD} and can occur at any point during the execution of REAct. For all simulations, we varied the rate at which the misbehaving node n_M repositions itself in P_{SD}. This repositioning, referred to as the average time between path modifications, is normally distributed and is represented in number of packets. We vary this average time between path modification from one audit duration to 3.5 audit durations, i.e., 200 packets to 700 packets. The variance is fix at 50 packets.

We assumed that the source is aware of any changes made to P_{SD}. If n_M moves to a different position in P_{SD}, but stays inside of the suspicious set $\mathcal{V}$, then no change is necessary in the searching process since this is analogous to n_M being in that position from the start of the search. The source will continue to execute the auditing until $|\mathcal{V}| = 2$. Suppose n_M moves to a position outside the set $\mathcal{V}$. To make sure that no node is excluded from $|\mathcal{V}|$ due to mobility, the source expands the suspicious set such that n_M is in $\mathcal{V}$. Thus, if $\mathcal{V} = \{n_i, \ldots, n_M, \ldots, n_j\}$ and n_M moves upstream of n_i, then the source will adjust $\mathcal{V}$ to $\{n_M, \ldots, n_i, \ldots, n_j\}$. A similar adjustment will be made if n_M moves downstream of n_j. All simulations are performed on a path of 16 nodes. We utilized two mobility models for the misbehaving node:

- **Normally Distributed Mobility**: Under this mobility model, n_M moves to a different position in P_{SD} according to a normal distribution. We set the average number of hops, μ_n, to 1 hop from its current position with a variance, σ_n of 1 hop. These numerical values were selected based on the actual limitations of a mobile node. Assume a radio range of 100 meters, a transmission speed of 1 Mbps, a packet size of 1500 bytes, and an audit duration a_d of 200 packets. For these parameters, auditing a single node requires 2.4 seconds. Given that a mobile node would have to travel at least 100 meters (the radio range) to move to a different position in P_{SD}, and varying the time between path modifications, this yields mobility speeds of $[150, 43]$ km/hr for the misbehaving node.

- **Uniformly Distributed Mobility**: Under this mobility model, n_M selects a different position in P_{SD} to move according to a uniform distribution. By fixing the mobility speed of n_M at 25 km/hr and varying the time between path modifications, we can determine the range data rates. For a misbehaving node that moves 1 hop the corresponding range of data rates is $[88.3, 291.7]$ kbps, while a misbehaving node that moves 16 hops yields the corresponding range of data rates is $[5.2, 18.2]$ kbps.

In Figure 5.3(a), we show the impact of node mobility on the communication overhead as a function of the average time between path modifications. Node n_M moves upstream or downstream according to a normal distribution. As expected, the less mobile the node is, the less overhead required to identify n_M. The overhead of REAct-B is independent of mobility. This is because it uses a batch auditing algorithm and thus does not rely on reducing the size of a suspicious set. All that is required is the identification of two misbehaving links. Note that regardless of the position of n_M, it is always identified in a misbehaving link. Additionally, if n_M moves, this is allows the source to identify a second misbehaving link without

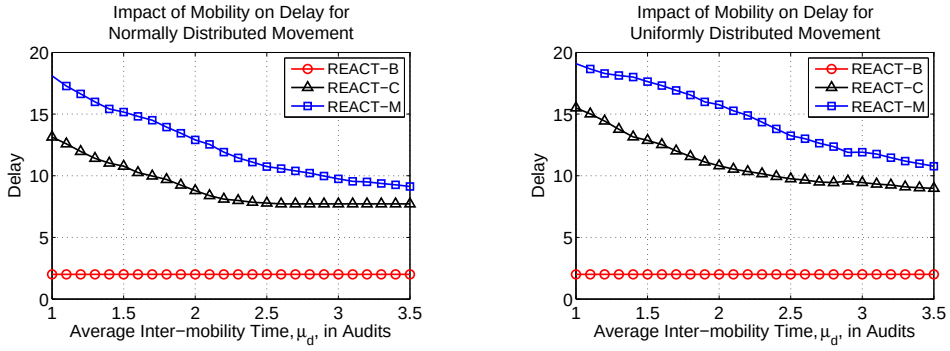


(a) (b)

Figure 5.3: (a) Impact of mobility on communication overhead as a function of the average time between path modifications. Communication overhead is shown as a normalized metric in which the transmission of a packet incurs a cost of 1 while the reception incurs the cost of 0.5. The misbehaving node repositions itself upstream or downstream according to a normal distribution with $\mu_n = 1$ hop and $\sigma_n = 1$. Path length is fixed at 16 nodes. (b) Impact of mobility on communication overhead as a function of the average time between path modifications. The misbehaving node repositions itself on P_{SD} randomly according to a uniform distribution.

requiring path division or path expansion, since the goal of each of these methods are to remove original misbehaving link from P_{SD}, which was accomplished during the movement of n_M. Thus the mobility of n_M does not provide any benefit with respect to communication overhead. This is not the case with REAct-C and REAct-M, since the auditing strategy is adaptive and thus continually reduce the set $\mathcal{V}$. Note that as the misbehaving node becomes less mobile, the communication overhead of REAct-C and REAct-M converges to the static P_{SD} case as shown in Figure 5.1(a).

In Figure 5.3(b), we show the impact of node mobility on the communication overhead as a function of the average time between path modifications, for uniformly distributed mobility. We note that under this model, the communication overhead for REAct-C and REAct-M is higher compared to normally distributed mobility. This is due to the fact that in the earlier case, a misbehaving node most likely remains in the suspicious set with no further impact on the identification process. On the other hand, under the uniform model, the suspicious set $\mathcal{V}$ is more likely to



(a) (b)

Figure 5.4: (a) Impact of mobility on communication overhead as a function of the average time between path modifications. Communication overhead is shown as a normalized metric in which the transmission of a packet incurs a cost of 1 while the reception incurs the cost of 0.5. The misbehaving node repositions itself upstream or downstream according to a normal distribution with $\mu_n = 1$ hop and $\sigma_n = 1$. Path length is fixed at 16 nodes. (b) Impact of mobility on communication overhead as a function of the average time between path modifications. The misbehaving node repositions itself on P_{SD} randomly according to a uniform distribution.

expand causing additional overhead until $|\mathcal{V}| = 2$. The performance of REAct-B is the same under both mobility models. This is expected, since REAct-B uses a batch auditing strategy and does not rely on reducing the suspicious set. As expected, the results for the identification delay converge to the state P_{SD} case shown in Figure 5.1(a) as mobility decreases.

In Figure 5.4(a), we show the impact of node mobility on the identification delay as a function of the average time between path modifications, under the normally distributed mobility model. Again, the identification delay for REAct-B is independent of the mobility of n_M. This is based of the same argument as before; namely that a misbehaving link is always identified in batch auditing mode, and thus two rounds of auditing, and hence a delay of 2, are required to identify the misbehaving node.REAct-C and REAct-M require an increased delay when n_M is highly mobile, due to the fluctuations in the suspicious set.

In Figure 5.4(b), we show the impact of mobility on the identification delay as a

function of the average time between path modifications, for a uniformly distributed mobility model. This model yields a higher delay for REAct-C and REAct-M since it is less likely for n_M to remain in the suspicious set when it moves. As with the communication overhead, REAct-B performs the same under both the normally distributed mobility model and the uniformly distributed mobility model. Again, note that both Figure 5.4(a) and Figure 5.4(b) converge to the state P_{SD} case shown in Figure 5.2(a).

5.3 Comparison with Other Schemes

We now compare the performance of our Rényi-Ulam inspired schemes to CONFIDANT [9], 2ACK [40], and AWERBUCH [2]. For 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 one [20]. For the 2ACK scheme, a fraction p of the messages transmitted by each node was acknowledged two hops upstream of the receiving node. We set that fraction to $p = \{1, 0.5, 0.1\}$ [40]. AWERBUCH identifies misbehaving links by probing the path, requiring selected nodes to send acknowledgment messages back to the source.

Since both CONFIDANT and 2ACK schemes are proactive, they incur communication overhead regardless of the existence of a misbehaving node. On the other hand, our schemes and AWERBUCH incur overhead only if misbehavior is observed, due to their reactive nature. We select the adaptive auditing strategy utilizing cut questions (REAct-C), for our figures. The plots in Figure 5.1(a)-(d) can be used for comparisons with REAct-B and REAct-M. To provide a fair comparison between the four schemes, we first considered the overhead during a fixed duration of time, i.e., the time required to identify the misbehaving node using REAct-C. The audit duration was set to 200 packets.

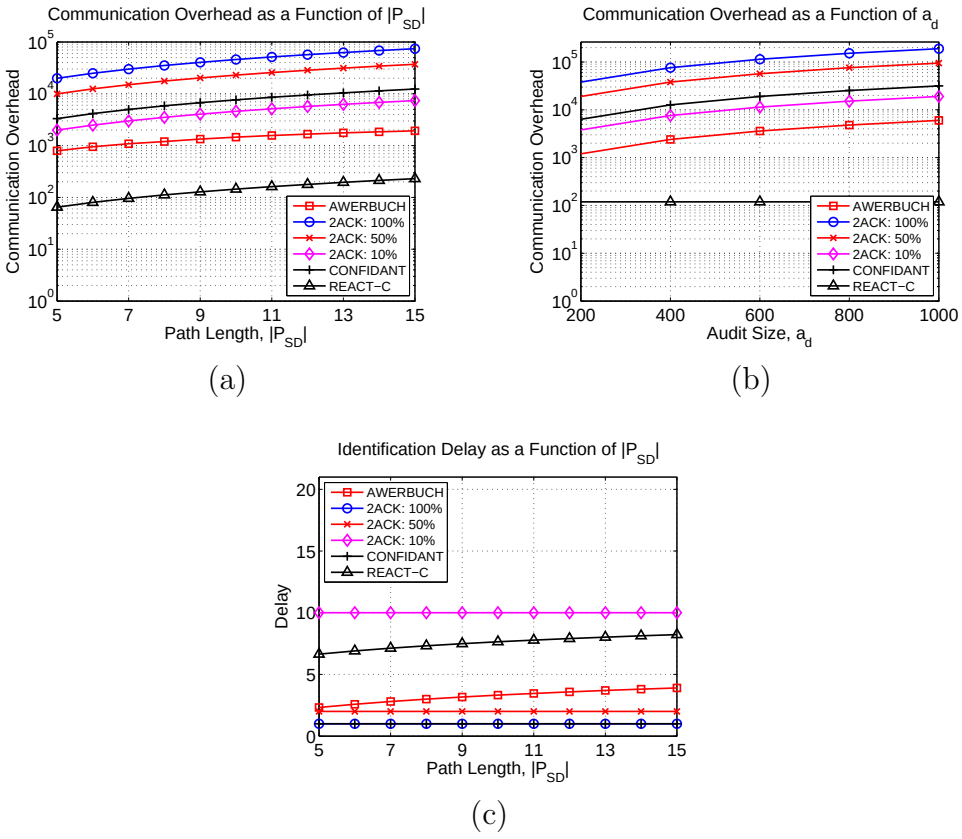


Figure 5.5: (a) Communication overhead as a function of path length, for an audit size of 200 packets ($a_d = 200$). The overhead is computed for the time required by REAct-C to converge on the misbehaving node. (b) Communication overhead as a function of audit size for a path length of eight nodes ($|P_{SD}| = 8$). (c) Identification delay as a function of the path length, for an audit size of 200 packets ($a_d = 200$).

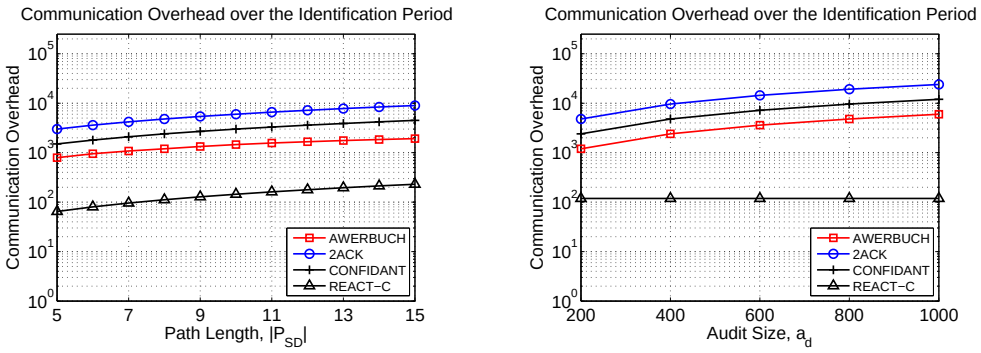
5.3.1 Fixed Time Communication Overhead

In Figure 5.5(a), we show the communication overhead as a function of path length. The Y axis is shown in logarithmic scale. The communication overhead for REAct-C is between 1-2 orders of magnitude less than the communication overhead for the other schemes. This gain is due to the fact that REAct-C does not expend energy on a per-packet basis to monitor the behavior of each node. As expected, AWERBUCH requires the next least amount of overhead, since it begins by probing one node,

then proceeding to probe one additional node each stage until the binary search terminates. The 2ACK scheme with $p = 1$ requires the highest communication overhead since every packet sent by the source requires a 2-hop acknowledgment to be sent upstream per link traversed. The communication overhead for REAct-M is in the same range as REAct-C, which can be shown since REAct-C and REAct-M require similar communication overhead as shown in Figure 5.1(a). Likewise, REAct-B requires approximately an order of magnitude less in communication overhead compared to 2ACK, CONFIDANT, and AWERBUCH.

The audit duration a_d defines the minimum number of packets that need to be monitored in order to differentiate normal packet loss from misbehavior. For example, when the audit duration is equal to 200 packets, a node's behavior must be monitored for at least 200 packets to decide whether its dropping rate is normal or constitutes misbehavior. In Figure 5.5(b), we show the communication overhead as a function of audit size for a path length of eight nodes. 2ACK, CONFIDANT, and AWERBUCH all incur a linear increase in communication overhead with the audit duration. This is due to the fact that for reputation- and acknowledgment-based methods, the communication overhead incurs on a per-packet basis. On the other hand, the communication overhead for REAct-C, REAct-M, and REAct-B is incurred on a per-audit basis. Thus the overhead of our algorithms is independent of audit duration.

While our algorithms provide significant savings in communication overhead, they require a longer time to identify the misbehaving node, since multiple audits need to be performed. On the other hand, the proactive schemes require only the duration of one audit to identify misbehavior. This is due to the fact that proactive protocols monitor all nodes in the path P_{SD} in parallel. Fortunately, for our adaptive auditing strategies REAct-C and REAct-M, the delay grows logarithmically with the path length. Hence, the increase in identification delay is small compared to the savings in communication overhead.



(a) (b)

Figure 5.6: (a) Communication overhead as a function of path length, $|P_{SD}|$, for an audit size of 200 packets ($a_d = 200$). For each scheme, the overhead is computed for the time required to identify the misbehavior. (b) Communication overhead as a function of audit size for a path length of eight nodes ($|P_{SD}| = 8$).

In Figure 5.5(c), we show the identification delay as a function of path length. CONFIDANT requires a single audit duration to identify the misbehaving node since all nodes in P_{SD} are monitored in parallel. AWERBUCH performs a binary search, thus requiring delay to increase logarithmically. The 2ACK scheme also requires a single audit duration for identification when all packets are acknowledged two hops upstream. However, the identification delay increases when only a fraction of the packets in one audit are acknowledged. For example, REAct-C achieves a delay metric comparable, though less, than 2ACK when 2ACK acknowledges only 10% of the packets. However, as shown in Figure 5.5(a), the communication overhead for REAct-C is an order of magnitude less.

5.4 Comparison Based on Identification Delay

According to Figure 5.5(c), the four compared schemes incur different delay during the misbehavior identification process. We now evaluate the communication overhead incurred by each scheme from the start of the node misbehavior until the identification of the misbehaving node. The communication overhead of CONFI-

DANT and 2ACK is measured for the duration of a single audit since all nodes are monitored in parallel, thus requiring only one audit duration for identification. Meanwhile, REAct-C and AWERBUCH are measured for the duration until the misbehaving node is identified.

In Figure 5.6(a), we show the communication overhead as a function of the path length, for an audit size of 200 packets. In Figure 5.6(b), we show the communication overhead as a function of the audit size, for a path length of eight nodes. We observe that even in the case where the communication overhead is measured only during the identification time, REAct-C significantly outperforms the other schemes, requiring at least one order of magnitude less in communication overhead. Both the CONFIDANT and 2ACK schemes are sensitive to path length and audit size. AWERBUCH, while logarithmic in its search, requires resource intensive acknowledgment messages. On the other hand, REAct-C allows for a graceful tradeoff between communication overhead and delay.

CHAPTER 6

Conclusions

Wireless ad hoc networks rely on multi-hop routes to transport data from a source to a destination, thus implementing the routing function in a collaborative function. A misbehaving node may refuse to forward packets to the next hop in order to conserve its energy resources (selfishness) or degrade network performance (maliciousness). In this thesis we investigated the problem of uniquely identifying the set of misbehaving nodes on which packet forwarding fails. We focused on improving the resource efficiency of the misbehavior identification mechanism.

We proposed a reactive method in which the source audits several nodes along the routing path to the destination, when notified of a performance drop. The purpose of these audits is to have nodes commit to the set of packets they forwarded to the next hop. The source collects multiple audit replies, combining them in order to identify the link on which packet forwarding fails. This is accomplished by progressively limiting a set of suspicious nodes, in which nodes upstream from the misbehaving nodes report forwarding all packets while nodes downstream from the misbehaving node report forwarding no packets.

When audited, transmission of the entire set of forwarded packets back to the source requires high communication and storage requirements. Thus we utilize Bloom filters to represent the set of forwarded packets in a compact manner. This provides two main advantages. The first is that the storage requirement for the Bloom filter is orders of magnitude less then storing and communicating all forwarded packets back to the source. Secondly, Bloom filters allow the explicit evaluation of the behavior of an audited node on a per-packet basis, while not incurring per-packet overhead. Thus we showed that the communication overhead for au-

diting is independent of the audit size, which is not true for acknowledgment- and reputation-based systems.

We showed that the collection and combination of audit replies to identify the misbehaving nodes is analogous to Rényi-Ulam games. Rényi-Ulam games are searching games where a responder selects a secret value in a finite search space which the questioner attempts to determine within a set number of questions. A unique trait of these games is the ability of the responder to lie up to a fixed number of times when answering questions. In the misbehavior identification problem, honest nodes always respond faithfully when audited, while a misbehaving node can lie. We employ efficient questioning strategies used in Rényi-Ulam games to identify nodes that lie about the set of packets forwarded to the next hop. Since our mapping collapses all misbehaving nodes to a single entity, collusion among misbehaving nodes is implicitly assumed.

Utilizing our mapping, we developed REAct, a resource-efficient misbehavior identification scheme. REAct is based on three variants of Rényi-Ulam games, thus providing the source with three different auditing strategies, one batch auditing strategy and two adaptive ones. In the batch auditing strategy called REAct-B, the source audits all nodes simultaneously. The source can always identify the misbehaving links, regardless of the number of misbehaving nodes. However, the required overhead increases linearly with the path length. Thus we examine adaptive auditing strategies in which the source adaptively selects nodes for audit based on past replies.

Our first adaptive auditing strategy was REAct-C, which was mapped from a cut questioning strategy proposed by Pelc [51]. This auditing strategy was based upon binary search. The source search the path until a link is identified, as which time an additional audit occurs to verify the result. This additional audit allows the source to identify if multiple misbehaving nodes exists, and thus to adapt its auditing strategy accordingly. In our second adaptive auditing strategy, REAct-M,

we mapped our solution to the membership questioning strategy proposed by Dhagat [17]. Using this method, the source searches for contradictions between current and past audit replies to identify the misbehaving link. We showed through simulation that although REAct-M results in greater delay than REAct-C, it provides increased savings in communication overhead when lies occur. Additionally, while REAct-C and REAct-M achieve a logarithmic increase in overhead, REAct-B requires the least delay since all audits are performed at once.

With the location of the misbehaving link, the source identifies the misbehaving node by isolating its two adjacent nodes. We presented two methods for the isolation, path division and path expansion. Both methods require the source to make a slight alteration in the routing path, effectively removing the previously discovered misbehaving link from the path. By re-executing the auditing algorithm, the source can identify the misbehaving node. It is this requirement of path modification that requires that the network be k-connected.

We compared REAct to two acknowledgment-based schemes [9, 40] and one reputation-based scheme in terns of communication overhead and identification delay. REAct provides at least an order of magnitude reduction in communication overhead. This overhead is independent of the amount of time that a node must be monitored to evaluate its behavior, i.e., the audit duration. On the other hand, the communication overhead increases linearly with monitoring duration for both acknowledgment-based and reputation-based systems. This is a desirable tradeoff given the relatively small path lengths observed in ad hoc networks.

REFERENCES

[1] R. Anderson and M. Kuhn. Tamper resistance-a cautionary note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, volume 2, pages 1–11, 1996.

[2] B. Awerbuch, D. Holmer, C.-N. Rotaru, and H. Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *In Proceedings of the ACM Workshop on Wireless Security (WiSe'02)*, 2002.

[3] K. Balakrishnan, J. Deng, and P. K. Varshney. Twoack: Preventing selfishness in mobile ad hoc networks. In *In Proceedings of IEEE Wireless Communications and Networking Conference (WCNC'05)*, 2005.

[4] E. Berlekamp. *Error Correcting Codes*. Wiley, N. Y., 1968.

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

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

[7] R. Brooks, P. Ramanathan, and A. Sayeed. Distributed target classification and tracking in sensor networks. *Proceedings of the IEEE*, 91(8):1163–1171, 2003.

[8] S. Buchegger and J.-Y. L. Boudec. Nodes bearing grudges: Towards routing security, fairness, and robustness in mobile ad hoc networks. In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP'02)*, pages 403–410, 2002.

[9] S. Buchegger and J.-Y. L. Boudec. Performance analysis of the confidant protocol (cooperation of nodes: Fairness in dynamic ad-hoc networks). In *Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC 2002)*, June 2002.

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

[11] S. Buchegger and J. Le Boudec. The effect of rumor spreading in reputation systems for mobile ad-hoc networks. In *Proceedings of WiOpt 03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, volume 4, pages 4–1, 2003.

[12] L. Buttyan and J. Hubaux. Nuglets: a virtual currency to stimulate cooperation in self-organized mobile ad hoc networks. *ICCA, Swiss Federal Institute of Technology*, 2001.

[13] L. Buttyan and J.-P. Hubaux. Enforcing service availability in mobile ad-hoc wans. In *Proceedings of the First Annual Workshop on Mobile and Ad Hoc Networking and Computing (MobiHOC'00)*, pages 87–96, 2000.

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

[15] J. Crowcroft, R. Gibbens, F. Kelly, and S. Östring. Modelling incentives for collaboration in mobile ad hoc networks. In *Proceedings of WiOpt03*, 2003.

[16] B. Culpepper, H. Tseng, N. Center, and C. Moffett Field. Sinkhole intrusion indicators in DSR MANETs. In *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, pages 681–688, 2004.

[17] A. Dhagat, P. Gács, and P. Winkler. On playing "twenty questions" with a liar. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 16–22, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[18] Y. Dong, H. Go, A. Sui, V. Li, L. Hui, and S. Yiu. Providing Distributed Certificate Authority Service in Mobile Ad Hoc Networks. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks 2005 (SecureComm 2005)*, pages 149–156, 2005.

[19] 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.

[20] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an ad hoc networking environment. In *Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '01)*, volume 3, 2001.

[21] S. Ganeriwal, L. Balzano, and M. Srivastava. Reputation-based framework for high integrity sensor networks. 2008.

[22] V. Gligor. Handling new adversaries in secure mobile ad-hoc networks. In *ARO Planning Workshop on Embedded Systems and Network Security (ESNS '07)*, 2007.

[23] N. Hanusse, E. Kranakis, and D. Krizanc. Searching with mobile agents in networks with liars. *Discrete Applied Mathematics*, 137(1):69–85, 2004.

[24] T. He, S. Krishnamurthy, J. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. Energy-efficient surveillance system using wireless sensor networks. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 270–283. ACM New York, NY, USA, 2004.

[25] L. Hu and D. Evans. Using directional antennas to prevent wormhole attacks. In *Network and Distributed System Security Symposium (NDSS)*, pages 131–141, 2004.

[26] Y. Hu, D. Johnson, and A. Perrig. SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks. *Ad Hoc Networks*, 1(1):175–192, 2003.

[27] Y. Hu, A. Perrig, and D. Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3.

[28] Y. Hu, A. Perrig, and D. Johnson. Rushing attacks and defense in wireless ad hoc network routing protocols. In *Proceedings of the 2nd ACM workshop on Wireless security*, pages 30–40, 2003.

[29] E. Huang, J. Crowcroft, and I. Wassell. Rethinking incentives for mobile ad hoc networks. In *Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, pages 191–196. ACM New York, NY, USA, 2004.

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

[31] 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.

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

[33] A. Kaporis, L. Kirousis, E. Kranakis, D. Krizanc, Y. Stamatiou, and E. Stavropoulos. Locating information with uncertainty in fully interconnected networks with applications to World Wide Web information retrieval. *The Computer Journal*, 44(4):221–229, 2001.

[34] L. Kirousis, E. Kranakis, D. Krizanc, and Y. Stamatiou. Locating information with uncertainty in fully interconnected networks. *Lecture notes in computer science*, pages 283–296, 2000.

[35] G. Kortuem, J. Schneider, D. Preuitt, T. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In *Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*, page 75. IEEE Computer Society Washington, DC, USA, 2001.

[36] E. Kranakis and D. Krizanc. Searching with uncertainty. In *Proc. 6th Int. Colloq. on Structural Information and Communication Complexity (SIROCCO99)*, pages 194–203, 1999.

[37] S. Kurosawa, H. Nakayama, N. Kato, A. Jamalipour, and Y. Nemoto. Detecting blackhole attack on AODV-based mobile ad hoc networks by dynamic learning method. *International Journal of Network Security*, 5(3):338–346, 2007.

[38] L. Lazos, R. Poovendran, C. Meadows, P. Syverson, and L. Chang. Preventing wormhole attacks on wireless ad hoc networks: a graph theoretic approach. In *2005 IEEE Wireless Communications and Networking Conference*, volume 2, 2005.

[39] A. Liu and P. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *to appear in Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*. SPOTS Track, 2008.

[40] 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 2007.

[41] Y. Liu and Y. R. Yang. Reputation propagation and agreement in mobile ad-hoc networks. In *Proc. of IEEE Wireless Communication and Networking Conference (WCNC'03)*, pages 1510–1515, March 2003.

[42] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, 2002.

[43] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. *Organization co-chairs*, 2004.

[44] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 255–265, 2000.

[45] P. Michiardi and R. Molva. Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *In Proceedings of the Sixth IFIP Conference on Security Communications and Multimedia (CMS02)*, 2002.

[46] S. Midkiff and C. Bostian. Rapidly-deployable broadband wireless networks for disaster and emergency response. In *Presented at The First IEEE Workshop on Disaster Recovery Networks (DIREN 02)*, 2002.

[47] E. Ngai, J. Liu, and M. Lyu. On the intruder detection for sinkhole attack in wireless sensor networks. In *IEEE International Conference on Communication (ICC)*, 2006.

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

[49] P. Papadimitratos and Z. Haas. Secure routing for mobile ad hoc networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, pages 01–27, 2002.

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

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

[52] A. Pelc. Fault-tolerant broadcasting and gossiping in communication networks. *Networks*, 28(3), 1996.

[53] A. Pelc. Searching games with errorsfifty years of coping with liars. *Theoretical Computer Science*, 270(1-2):71–109, 2002.

[54] C. Perkins, E. Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing, 2003.

[55] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8(5):521–534, 2002.

[56] A. Pirzada and C. McDonald. Circumventing sinkholes and wormholes in wireless sensor networks. In *International Conference on Wireless Ad Hoc Networks (IWWAN)*, 2005.

[57] R. Poovendran and L. Lazos. A graph theoretic framework for preventing the wormhole attack in wireless ad hoc networks. *Wireless Networks*, 13(1):27–59, 2007.

[58] S. Raghani, D. Toshniwal, and R. Joshi. Dynamic Support for Distributed Certification Authority in Mobile Ad Hoc Networks. In *Proceedings of the 2006 International Conference on Hybrid Information Technology-Volume 01*, pages 424–432. IEEE Computer Society Washington, DC, USA, 2006.

[59] Y. Rebahi, V. Mujica, and D. Sisalem. A reputation-based trust mechanism for ad hoc networks. In *In Proceedings of the 10th IEEE Symposium on Computers and Communications (ISCC05)*, pages 37–42, 2005.

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

[61] R. Rivest, A. Meyer, D. Kleitman, K. Winklmann, and J. Spencer. Coping with errors in binary search procedures. *J. Comput. System Sci*, 20:396–404, 1980.

[62] N. Salem, L. Buttyán, J. Hubaux, and M. Jakobsson. A charging and rewarding scheme for packet forwarding in multi-hop cellular networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 13–24. ACM New York, NY, USA, 2003.

[63] S. Soltanali, S. Pirahesh, S. Niksefat, and M. Sabaei. An Efficient Scheme to Motivate Cooperation in Mobile Ad hoc Networks. In *Networking and Services, 2007. ICNS. Third International Conference on*, pages 98–98, 2007.

[64] J. Spencer and P. Winkler. Three thresholds for a liar. *Combinatorics, Probability and Computing*, 1:81–93, 1992.

[65] L. Tamilselvan and V. Sankaranarayanan. Prevention of Blackhole Attack in MANET. In *Wireless Broadband and Ultra Wideband Communications, 2007. AusWireless 2007. The 2nd International Conference on*, pages 21–21, 2007.

[66] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.

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

[68] 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.

[69] S. Yi and R. Kravets. MOCA: Mobile Certificate Authority for Wireless Ad Hoc Networks. In *2nd Annual PKI Research Workshop Pre-Proceedings*, volume 51, page 61801.

[70] S. Yousefi, M. Mousavi, and M. Fathy. Vehicular ad hoc networks (VANETs): challenges and perspectives. In *ITS Telecommunications Proceedings, 2006 6th International Conference on*, pages 761–766, 2006.

[71] J. Zhao and G. Cao. VADD: Vehicle-assisted data delivery in vehicular ad hoc networks. *IEEE Transactions on Vehicular Technology*, 57(3):1910–1922, 2008.

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

[73] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 2005.