UNMASK: Utilizing Neighbor Monitoring for Attack Mitigation in Multihop Wireless Sensor Networks

Issa Khalil
College of Information Technology (CIT)
United Arab Emirates University (UAEU)
Email: ikhalil@uaeu.ac.ae

Saurabh Bagchi, Cristina N.-Rotaru, Ness Shroff
School of Electrical& Computer Engineering and Dep. of Computer Science, Purdue University
Email: {sbagchi, crisn, shroff}@purdue.edu

Corresponding Author: Issa Khalil {ikhalil@uaeu.ac.ae}

Abstract

Sensor networks enable a wide range of applications in both military and civilian domains. However, the deployment scenarios, the functionality requirements, and the limited capabilities of these networks expose them to a wide-range of attacks against control traffic (such as wormholes, rushing, Sybil attacks, etc) and data traffic (such as selective forwarding). In this paper we propose a framework called UNMASK that mitigates such attacks by detecting, diagnosing, and isolating the malicious nodes. UNMASK uses as a fundamental building block the ability of a node to oversee its neighboring nodes' communication. On top of UNMASK, we build a secure routing protocol, LSR, that provides additional protection against malicious nodes by supporting multiple node-disjoint paths. We analyze the security guarantees of UNMASK and use ns-2 simulations to show its effectiveness against representative control and data attacks. The overhead analysis we present shows that UNMASK is a lightweight protocol appropriate for securing resource constrained sensor networks.

Keywords: sensor network security, neighbor monitoring, secure routing, control attack, data attack.

1 Introduction

Wireless sensor networks are emerging as a promising platform that enable a wide range of applications in both military and civilian domains such as battlefield surveillance, medical monitoring, biological detection, home security, smart spaces, inventory tracking, etc. Such networks consist of small, low-cost, resource-limited (battery, bandwidth, CPU, memory) nodes that communicate wirelessly and cooperate to forward data in a multi-hop fashion. Thus, they are especially attractive in scenarios where it is infeasible or expensive to deploy a significant networking infrastructure. However, the open nature of the wireless communication, the lack of infrastructure, the fast deployment practices, and the hostile deployment environments, make sensor networks vulnerable to a wide range of security attacks targeting the control or data traffic. Coping with control and data attacks in sensor networks is more challenging than in ad hoc wireless and wired networks due to the resource constrained environment.

Typical examples of control traffic (numbers start with C for control) are routing, monitoring whether a node is awake, asleep, or dead, topology discovery, and distributed location determination. Control traffic attacks include the (Ci) wormhole attack ([16],[17]), (Cii) the rushing attack [18], (Ciii) the Sybil attack [11], (Civ) the sinkhole attack [14], and (Cv) the HELLO flood attack [14]. Control attacks are especially dangerous because they can be used to subvert the functionality of the routing protocol and create opportunities for a malicious node to launch data traffic attacks such as dropping all or a selective subset of data packets.

In addition to control traffic attacks, sensor networks are also vulnerable to data traffic attacks (numbers starts with D for data). The most notable data traffic attacks are (Di) blackhole, (Dii) selective forwarding and (Diii) artificially delaying of packets, in which respectively a malicious node drops data (entirely or selectively) passing through it, or delays its forwarding. The attacks could result in a significant loss of data or degradation of service.

The focus of this paper is on proposing mitigation techniques for control and data attacks in sensor networks. We present a lightweight framework called UNMASK (**U**tilizing **N**eighbor **M**onitoring for **A**ttacks Mitigation in Multihop Wireless **S**ensor Networ**k**s), which mitigates control and data traffic attacks in sensor networks. UNMASK not only detects the occurrence of an attack, but also diagnoses the malicious nodes involved in it and removes their capability of launching future attacks by isolating them from the network. The detection and isolation mechanisms are executed locally, without incurring a significant overhead. UNMASK is suited to the low cost point of sensor networks since it does not require any specialized hardware (such as directional antennas [17] or GPS) nor does it require time-synchronization among the nodes [16]. UNMASK achieves its security goals by exploiting a well-known technique whereby nodes oversee part of the traffic going in and out of their neighbors [15], [25], [30], [31]. In our work, we present the technique in a formal framework–*local monitoring*–, identify the parameters that affect its performance, and analyze its capabilities and limitations. UNMASK can be applied to mitigate any control or data traffic attack that exploits one or more of the basic malicious primitives- drop, delay, fabricate, and modify. However, we exemplify the fundamental structures and the state to be maintained at each node for mitigating some representative attacks – Sybil, wormhole, rushing, and selective forwarding attacks. The first three are examples of attacks directed to control traffic while the last one is an example directed at data traffic. Independent of the detection mechanism, we propose a strategy to isolate malicious nodes locally in a distributed manner.

We use UNMASK to create a novel lightweight secure routing protocol called LSR that withstands known attacks against the routing infrastructure and provides additional protection against data attacks by supporting secure node-disjoint multiple route discovery. We analyze the detection coverage and the probability of false detection of UNMASK. We also evaluate the memory, communication, and computation overhead of UNMASK. Finally, we simulate the wormhole attack in *ns-2* and show its effect on the network performance with and without UNMASK. The results show that UNMASK can achieve 100% detection of the wormholes for a wide range of network densities. They also show that the detection and isolation of the nodes involved in the wormhole can be achieved in a fairly short time after an attack starts. In addition, we simulate a combined Sybil and rushing attack to bring out the adverse impact on node-disjoint multipath routing and show the improvement using UNMASK. The results show that LSR using UNMASK is resilient to the combined attack and that the average number of node-disjoint routes discovered is not reduced. Our experiments with data monitoring show the feasibility of detecting the selective forwarding attack while monitoring only a fraction of the data traffic.

In summary, our contributions are as follows:

- We propose a mechanism to detect any control or data attack that results from dropping, delaying, modifying, or fabricating of packets.

- We develop a toolset based on overheard information that can be mapped to detecting different classes of attacks. We analyze this toolset for different metrics, such as, false alarm probability, missed alarm probability, and latency of isolation.
- We propose a mechanism that, based on information collected by our toolset, allows for diagnosing and isolating the malicious nodes.
- We demonstrate the effectiveness of our toolset applied to both data and control attacks through simulations.

This work builds on and extends our previous work in applying local monitoring as presented in [36] and [37]. Details about the differences are presented in Section 2.

The rest of the paper is organized as follows. Section 2 presents the related work in the area of security in wireless ad-hoc and sensor networks. Sections 3 and 4 describe UNMASK and LSR, respectively. Section 5 presents attacks against routing and their mitigation using LSR with UNMASK. Section 6 analyzes the coverage and overhead of UNMASK, while Section 7 shows simulation results. Section 8 concludes the paper.

2 Related Work

In the last few years, researchers have been actively exploring many mechanisms to ensure the security of control and data traffic in wireless networks. These mechanisms can be broadly categorized into the following classes – cryptographic building blocks used as support for key management, authentication and integrity services, protocols that rely on path diversity, protocols that overhear neighbor communication, protocols that use specialized hardware and protocols that require explicit acknowledgements or use statistical methods. The cryptographic primitives are also used as building blocks for protocols of the other classes.

In the context of ad hoc networks, HMAC and digital signatures [33] have been used to provide end-to-end authentication of the routing traffic [1],[4]. Intermediate node authentication of the source traffic has been achieved via broadcast authentication techniques using digital signatures [19], hash trees [2], or μ-TESLA [3]. One-way key chains and Merkle hash trees were also used as a defense against Sybil attacks [38]. These protocols are restrictive and only capable of providing basic security guarantees, namely confidentiality and authenticity of the control and data traffic, or address only a specific attack such as Sybil. In addition, these protocols are not appropriate for sensor networks since the public key cryptography is beyond the capabilities of sensor nodes and the symmetric key based protocols used in [2], [3], [19], and [38] are too expensive in terms of node state and communication overhead. A specific solution for the wormhole attack proposed in [42] uses keys known in a local region to prevent a message replayed by a malicious node from being decrypted at a distance. The solution uses specialized trusted nodes which cannot be affected by any wormhole.

The path diversity techniques increase route robustness by first discovering multi-path routes [19],[25],[26],[44] and then using these paths to provide redundancy in the data transmission between a source and a destination [24]. The data is encoded and divided into multiple shares sent to the destination via different routes. The method is effective in well-connected networks, but does not provide enough path diversity in sparse networks. Moreover, many of these schemes are expensive for sensor networks due to the data redundancy and are vulnerable to route discovery attacks, such as the Sybil attack, that prevent the discovery of non-adversarial paths.

Mechanisms to overhear neighbor communication in a wireless channel have been used to minimize the effect of misbehaving nodes [15],[26],[30]-[32]. One example is the watchdog scheme [15], where the sender of a packet watches the behavior of the next-hop node for that packet. If the next-hop node drops or tampers with the packet, the sender announces it as malicious to the rest of the network. The scheme is vulnerable to framing, does not work correctly when malicious nodes collude, and can have a high error-rate due to collisions in the wireless channel. Neighbor watch has also been used to build trust relationships among nodes in the network [30],[31], to build cooperative intrusion detection systems [32], or to discover multiple node-disjoint routes [26]. However, all these protocols use communication overhearing as an existing service without studying its feasibility, requirements, limitations, or performance in the resource-constrained sensor environment.

Examples of protection mechanisms that require specialized hardware are [16],[17],[40],[41]. The first scheme called *packet leashes* uses either tight time-synchronization or location awareness through GPS hardware, while the second uses directional antennas to detect wormhole attacks. The work in [40] relies on hardware threshold signature implementations to prevent one node from propagating errors or attacks in the whole network. In [41], the protocol uses locators with high powered directional antennas that broadcast beacons which are used by sensors to localize themselves.

Another technique proposed to detect malicious behavior that results in degradation of delivery ratio due to selective dropping of data, relies on explicit acknowledgement for received data using the same channel [44] , or out-of-band channel [43]. This method incurs high communication overhead which may be unsuitable for sensor networks and it has to be augmented by other techniques for diagnosis and isolation of the malicious nodes. A natural extension would be to reduce the control message overhead by reducing the frequency of ack-ing to one in every *N* data messages (in the above papers *N*=1). However, this may delay the adversary detection which may result in significant damage. [49] uses Neighbor Number Test (NNT) and All Distances Test (ADT) as statistical measures for inferring the existence of wormholes. The NNT detects the increase in the number of the neighbors of the sensors, which is due to the new links created by the wormhole in the network. The ADT detects the decrease of the lengths of the shortest paths between all pairs of sensors, which is due to the shortcut links created by the wormhole in the network. However, these algorithms are centralized and they have to be run by the base station on the network graph. The base station builds the network graph by collecting the neighbor lists from all the sensors in the network.

On the other hand, many sensor network routing protocols have also been introduced in the literature [5]-[9]. These protocols are less complex than ad hoc or wired routing protocols and are susceptible to a wide variety of attacks, as summarized by Karlof and Wagner [14]. Table 1 enumerates the protocols and their vulnerabilities.

Table 1: Attacks against wireless routing protocols (Numbers refer to the numbered list in the introduction)

Routing protocol name	**Attacks**
Directional diffusion ([5], [7])	Ciii, Civ, Cv, Dii
GPSR [6]	Ciii, Dii
Minimum cost forwarding [8]	Ci, Civ, Cv, Dii
LEACH [9], PEGASIS [20]	Cv, Dii
Rumor routing [10]	Ci, Ciii, Civ, Dii
SPAN [13]	Ciii, Cv

Few of the protocols mentioned discuss the method for removing the malicious nodes from causing further damage in the network and even fewer provide a quantitative analysis of the detection coverage, which may be affected due to a faulty detector or environmental conditions.

Our previous protocol called LITEWORP [36] introduced local monitoring and used it to address a specific control attack, called the wormhole attack. The follow-on work in [37] generalized the detection mechanism to detect a wider class of control attacks. However, the protocols did not focus on data attacks and did not address the issue of data traffic monitoring. This paper extends our previous work to address a wide class of control or data attacks in a unified framework and provides the corresponding checking mechanisms that can be used to detect each attack primitive as well as the corresponding overhead analysis.

3 Description of UNMASK

In its goal of providing detection and isolation to control and data attacks, UNMASK provides the following primitives - *neighbor discovery* and *one-hop source authentication* (Section 3.2). These two primitives are then used as the building blocks for the two main modules - *local monitoring* (Section 3.3) and *local response* (Section 3.5).

3.1 System Model and Assumptions

Attacker model: An attacker can control an external node (i.e., a node that does not know the cryptographic keys that allows it to be authenticated by the rest of the nodes), or an internal node, (i.e., a node that possesses all the keys required for it to be authenticated by other nodes in the network, but exhibits malicious behavior). An insider node may be created, for example, by compromising a legitimate node. A malicious node can perform all the attacks mentioned in Section 1, by itself or by colluding with other nodes. However, we do not address the misrouting attack in which the attacker incorrectly forwards packets nor we consider the denial of service attacks. A malicious node can establish out-of-band fast channels (e.g., a wired link) or have a high powered transmission capability.

***System assumptions**:* We assume that all the communication links are bi-directional. A finite amount of time is required from a node's deployment for it to be compromised, and to perform the first- and second-hop neighbor discovery protocol. We assume that no external or internal malicious nodes exist before the completion of the neighbor discovery. However, we can remove this assumption and use one of the protocols for secure neighbor discovery such as the directional antenna by Hu and Evans [17] at the additional cost of using directional antennas or by using trusted and more powerful nodes as in [42]. In our protocol we call the sensor node a *guard* when performing traffic overhearing and monitoring of neighbors. We assume that the network has sufficient redundancy, such that each node has more than an application defined threshold number of legitimate nodes as guards. We assume that the network has a static topology. This does not rule out route changes due to natural and malicious node failures or route evictions from the routing cache. Moreover, we assume that each node explicitly announces the immediate source of the packet it is forwarding. Finally, we assume a key management protocol, e.g., [22], is used to pre-distribute pair-wise keys such that any two nodes in the network can securely communicate with each other.

3.2 Primitives: Neighbor Discovery and One Hop Source Authentication

Neighbor discovery: This protocol is used to build a data structure of the first hop neighbors of each node and the neighbors of each neighbor. The data structure is used in local monitoring to detect malicious nodes and in local response to isolate these nodes. A neighbor of a node, W, is any node that lies within the transmission range of W. As soon as a node, say A, is deployed in the field, it sends a one-hop broadcast of a HELLO message. Any node that receives the message sends a reply back to A. For each reply received within a pre-defined timeout (T_{ROUT},), A adds the responder to its neighbor list, R_A. Let $R_A = W_1, ..., W_p$ and $M_{sg} = R_A||K_{commit(A)}$, where $K_{commit(A)}$ is the commitment key A uses later to authenticate itself to its neighbors. Node A then sends a one-hop broadcast of M_{sg}. A node W_j that receives M_{sg}, stores R_A (W_j's second-hop neighbors) and $K_{commit(A)}$. Hence, at the end of this neighbor discovery process, each node has a list of its direct neighbors and their neighbors as well as the commitment key of each one of its direct neighbors. This process is performed only once in the lifetime of a node and prevents incorrect neighborhood membership in static wireless networks that follow our assumptions of attack-free environment during neighbor discovery.

Commitment key generation and update: This protocol is used to generate and update the *commitment key* used by the *one-hop source authentication* protocol. The values of the commitment key at a node S ($K_{commit(S)}$) are derived from *a random seed* ($K_{seed(S)}$) as $K_{commit(S)} = H^{(i)} (K_{seed(S)})$, where H is a one-way collision resistant hash function [45]-[47], i takes values between 0 and $l(\geq 2)$, and l is the length of the sequence of values of $K_{commit(S)}$ that we call the *commitment string*. The first value of the commitment key $K_{commit(S)}$ that is exchanged with the neighbors during neighbor discovery is $H^{(l)}(K_{seed(S)}) = v_l$. The subsequent values of the commitment key ($v_{l-1},\ldots, v_0$) are progressively disclosed to the neighbors during subsequent transmissions. Before the current commitment string $\{v_l, v_{l-1},\ldots, v_0\}$ is exhausted, a new one is generated at S $\{u_l, u_{l-1},\ldots,u_0\}$. The commitment key u_l from the new string is authenticated to the neighbors using the last undisclosed key from the current string with the one-hop source authentication protocol.

One-hop source authentication: This protocol allows a node to distinguish between its neighbors to prevent identity spoofing among them. A node S authenticates its transmitted packets to the neighbors by attaching the last undisclosed value from the commitment string $K_{commit(S)}$. This authentication is only used with the source of the packet, not at every hop in the path of the packet from the source to the destination . When a neighbor of S, say B, receives the packet, it verifies the validity of $K_{commit(S)}$ by computing a hash function over it and comparing the result with the stored value of $K_{commit(S)}$. If $K_{commit(S)}$ is valid, B stores it as the new commitment key value of S. However, this protocol may fail to provide the required authentication if an attacker blocks the transmission range of a certain source from the rest of network except itself. Therefore, the attacker can impersonate that source and generate valid packets. In such case, we revert to the well-known μTESLA authentication scheme [21] which countermeasures such attacks.

3.3 Local Monitoring: Technique for Detection and Diagnosis of Attacks

This module detects various attacks against the control and data traffic and diagnoses the malicious nodes involved in the attacks. Local monitoring starts immediately after the completion of neighbor discovery. It uses a collaborative detection strategy, where a node monitors the traffic going in and out of its neighbors.

For a node, say W, to be able to monitor a node, say A, two conditions are required: (i) each node must explicitly announce the immediate source (i.e., the previous-hop relay node) of the packet it is forwarding, and (ii) W must be a neighbor of both A and the previous hop from A, say Y.

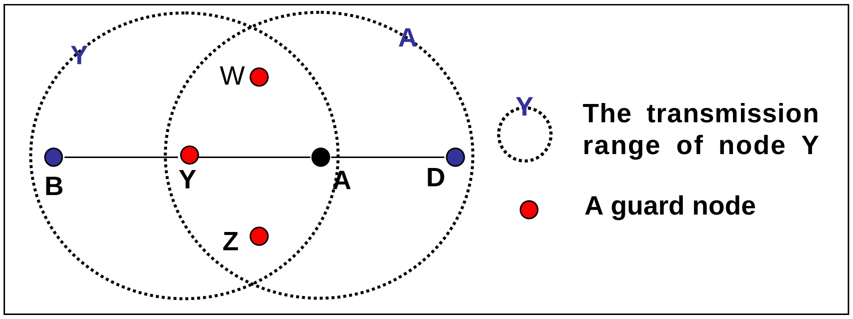


Figure 1: W, Y, and Z are guards of A over link Y to A

The first condition holds by design of the routing protocol and thus the second condition is the deciding criterion. In such a case, we call *W* a *guard* node of *A* over the link from *Y* to *A*. In Figure 1, nodes *W*, *Y*, and *Z* are the guards of *A* over the link from *Y* to *A*. For a link (*I*,*J*), the sender *I* is a guard node for node *J*. Information for each packet sent from *Y* to *A* is saved in a *watch buffer* at each guard for a time τ. The information maintained depends on the particular attack under consideration.

A malicious counter (*MalC(I,J)*) is maintained at each guard node, *I*, for every node, *J*, which *I* is monitoring. *MalC(I,J)* is incremented for any suspect malicious activity of *J* that is detected by *I*. To account for intermittent natural failures that can occur at legitimate nodes, a node is determined to be misbehaving, only if the *MalC* goes above a threshold.

In a general sense, the elementary activities underlying a large set of attacks in an ad-hoc multi-hop network are comprised of the following actions performed by the adversary node on an incoming packet – delay, drop, modify, and fabricate. There are elementary checking actions on the watch buffer for detecting each of these elementary malicious activities. The exact information stored in the watch buffer depends on the type of checking action – if delay, drop, or fabrication is to be detected, then only the header information that uniquely identifies the packet (in our implementation, the sender and the sequence number) need be stored. If however, modification to the payload is also to be detected, then a hash of the payload body has also to be stored. The actions are specified in Table 2. These checking actions form the basis of detection in UNMASK. In this paper, we discuss the detection for a representative set of attacks, though the elementary checking activities can be combined to detect a larger class of attacks.

Table 2: Elementary malicious activity and checking action

Elementary malicious activity	**Elementary checking action**
Delay	A packet lies unmatched in the buffer for time greater than an application dependant threshold.
Drop	Same as in delay.
Modify	The outgoing packet does not match with the packet in the watch buffer. The matching may be either a bit-wise comparison of the unchanging fields in the packet (such as, the data, the original source and destination) or matching the hash values computed on these fields.
Fabricate	The outgoing packet does not have a corresponding packet in the watch buffer.

3.4 Application of Local Monitoring for Data Attacks

We refer to data attacks as the general class of attacks directed at the data traffic after the route has been established. The objective of these attacks is to disrupt the end-to-end transmission of data between a source and a destination. The disruption can be done through leaking information or through launching denial of service by manipulating the data. When leaking information, the adversary node does not manipulate the data but gathers information based on data that flows through it. In the denial of service attack, the adversary actively manipulates the data packets through delay, drop, fabrication, or modification. Information leaking is difficult to detect by monitoring the data traffic alone. Information leaking becomes particularly insidious when the adversary uses control attacks such as the wormhole attack to create an opportunity to control a disproportionately large portion of the routes in the network. We use the local monitoring approach applied to the control traffic to prevent information leakage from exploiting control traffic attacks.

For the second type of data attacks (DoS by manipulating the data), local monitoring can be applied to the data traffic using the elementary checking activities mentioned in Table 2. This approach is useful in particular where an adversary node is in the position of having large amounts of data traffic flowing through it due to its strategic position in the network, without the need to launch a wormhole attack. The detection of data traffic manipulation in such a case can significantly improve the delivery ratio of the network.

In UNMASK, the guard node maintains in its watch buffer a data structure containing the following information about the observed packets: immediate source, immediate destination, original source, final destination, packet id (unique *wrt* a sender), and packet information. The packet information may be the unchanging fields in the packet header, the hash value of the unchanging fields in the header and the payload, or the entire packet itself. The elementary checking actions mentioned in Table 2 are performed on this information. The key distinction of data traffic monitoring from control traffic monitoring is the volume of traffic. Therefore, each guard node selects a fraction of the data traffic to monitor. In the current design, this is a global parameter for all the nodes. The fraction of traffic monitored is calculated over a given time window. Also for detecting modification, only hash values are matched, using a collision free yet computationally inexpensive hashing technique, such as SHA-1 [35].

3.5 Local Response and Isolation

Detection and diagnosis are only the first steps towards protecting the network. The local response and isolation module is used to propagate the detection knowledge to the neighbors of the malicious node and to take the appropriate response to isolate it from the network. Isolation in the context of this paper means removing the isolated node's capability of communicating with any other legitimate node in the network, thereby eliminating its potential for launching further instances of whichever attack we were dealing with in the first place.. The following local response algorithm is triggered by a guard node, say α, when a node, say A, is suspected because its *MalC* counter value crosses the threshold.

1. Node α removes A from its neighbor list, and sends to each neighbor of A, say W, an authenticated alert indicating that A is a suspected malicious node. The communication is authenticated using the shared key between α and W to prevent false accusations. This constitutes the *direct detection*.
2. When W receives the alert, it verifies its authenticity, that α is a neighbor to A, and that A is W's neighbor. It then stores ID_α in an *alert buffer* associated with A.
3. When W receives enough alerts, γ, about A, it isolates A by marking A's status as void in the neighbor list. We call γ the *detection confidence index*. This constitutes the *indirect detection*. The detection confidence represents the minimum number of guard nodes that must report that a certain node, j, is malicious for a neighbor, i, of j to isolate it, if i does not directly detect j. Note that the number of guards that report malicious activity is cumulative over time. A single node, due to the authentication mechanism, cannot generate more than one acceptable alert. Framing is the process by which an innocent node is proved to be malicious by a quorum of malicious nodes. A small value for γ increases the chance of successful framing of good nodes, while a large value of γ increases the delay before which a malicious node is locally detected and isolated. If we set γ to be infinity it means that a node only trusts itself in revoking a suspicious node and thus the local framing probability goes to zero. False alarm, distinct from framing, is caused by a (legitimate) node mistaking another (legitimate) node to be malicious because of imperfections in the wireless channel, such as collisions and losses. For example, node i may not observe node j dutifully forwarding a packet.
4. After isolating A, node W does not accept any packet from or forward any packet to A.

In addition to removing the malicious nodes from the network, this module makes the response process fast since the detection knowledge need not propagate throughout the network. This module is lightweight in the number of messages (one to each neighbor of A, only on detection) and the number of hops each message traverses (maximum two hops).

4 LSR: Lightweight Secure Routing

LSR is an on-demand routing protocol, sharing many similarities with the AODV [23] protocol. However, LSR has significant differences in order to enhance security. The design features of LSR described below make it resilient to a large class of control attacks such as wormhole, Sybil, and rushing attacks, as well as authentication and ID spoofing attacks. Combined with UNMASK, LSR can deterministically detect and isolate nodes involved in launching these attacks.

4.1 Route Discovery and Maintenance

Route Request: When a node, say *S*, needs to discover a route to a destination, say *D*, it generates a route discovery packet (*REQ*) that contains: a flag to indicate that it is a route request packet (F_{REQ}), the sender's identity (ID_S), the destination's identity (ID_D), and a unique sequence number (*SN*). The *SN* is incremented with every new *REQ* and is used to prevent the replay of the *REQ* packet. Node *S* then calculates a message authentication code (*MAC*) of the packet using the shared key between *S* and *D* (K_{SD}). Finally, *S* generates and attaches the next value of the commitment key $K_{commit(S)}$ to the *REQ* packet and broadcasts it.

1. [At *S*] $REQ = F_{REQ} \,||\, ID_S \,||\, ID_D \,||\, SN$
2. $S \xrightarrow{Broadcast} REQ||MAC_{K_{SD}}(REQ)||K_{commit(S)}||ID_S$

A neighbor *Z* of *S* accepts the *REQ* packet if the associated $K_{commit(S)}$ is valid. Then *Z* removes $K_{commit(S)}$ from the *REQ*, attaches ID_Z, and forwards the *REQ*.

An intermediate node *B* that is not a direct neighbor to *S* stores the first *REQ* packet it receives. Node *B* also keeps the identity of every different neighbor that forwards a subsequent copy of the same *REQ* during a *rush time*, T_r, selected randomly from [$T_{min,}$ T_{max}], as in [18]. When T_r runs out or when a certain *number of requests*, N_r, is collected, whichever occurs first, *B* broadcasts a randomly selected copy of the *REQ* copies that it has. Assume, without loss of generality, that *B* selects the one forwarded by *W*. For each source-destination pair, node *B* keeps the identity of the node from which it receives the forwarded *REQ* (ID_W). Node *B* then appends ID_B and ID_W to the *REQ* and broadcasts it. The process continues until the *REQ* reaches *D*.

3. $B \xrightarrow{Broadcast} REQ||MAC_{K_{SD}}(REQ)||ID_W||\, ID_B$

Route Reply: When *D* receives the *REQ* packet, it verifies the authenticity of the source using the shared key K_{SD}. Then *D* generates a route reply packet *REP* that contains: a flag to indicate that it is a route reply packet (F_{REP}), the sender identity (ID_S), the destination identity (ID_D), and a *SN*. Node *D* then calculates a MAC value over the packet using K_{SD}. Node *D* generates and attaches the next value of the commitment key $K_{commit(D)}$ to the *REP* packet. Finally, *D unicasts* the *REP* packet back to the previous hop as determined by the *REQ* packet. Let *A* be the immediate previous hop from *D* and *C* the immediate previous hop from *A*.

1. [At *D*] $REP = F_{REP}||ID_S||ID_D||\, SN$
2. $D \rightarrow A{:}REP||\, MAC_{K_{SD}}(REP)\, ||\, K_{commit(D)}||ID_D||ID_A$

When *A* receives the *REP* packet, it verifies and removes $K_{commit(D)}$, updates its routing table as follows - <Destination, Next-hop>: {*D*, *D*}, {*S*, *C*}. Node *A* then appends $ID_D||ID_A||ID_C$ and sends the *REP* packet to *C*.

3. [At *A*] Verify and remove $K_{commit(D)}$. Set <Destination, Next-hop>: {*D*, *D*}, {*S*, *C*}
4. $A \rightarrow C$: $REP||MAC_{K_{SD}}(REP)||\, ID_D \,||\, ID_A \,||\, ID_C$

The *REP* continues to propagate using the reverse path of the corresponding *REQ* towards *S*. Node *S* verifies the authenticity of the reply using K_{SD} and updates its routing table to node *D*.

The route maintenance in LSR, as in AODV, is triggered when a broken link is detected and a new route is discovered by using the above protocol for route discovery. Note that in LSR, the source chooses the route corresponding to the fastest route reply and not the shortest-hop route, to guard against attacks that modify the hop count. A longer but less congested route is preferred to a shorter but congested route, as in [19].

4.2 Node-Disjoint Multipath Discovery

A beneficial feature of LSR is its ability to increase the number of node-disjoint routes between a source and a destination. In many on demand ad-hoc and sensor network routing protocols, an intermediate node forwards the first announcement of a request and suppresses any following announcements, such as in AODV [23]. As a result, multiple routing paths may have common nodes in them. In LSR, each node, say *B*, backs off for a random time (T_r) before forwarding the *REQ*. During T_r, *B* buffers all the announcements of the same request. At the same time, *B* listens to any neighbor, say *E*, whose rush timer, T_r times out and which forwards one of its *REQ* copies. If *B* has the same *REQ* copy, from the same previous hop, as that forwarded by *E*, *B* deletes that copy from its buffer and thus will not be a candidate for *REQ* forwarding by *B*.

An example is shown in Figure 2. Let *B* receive *REQ*s from nodes *W*, *Y*, and *Z*, and let *E* be a neighbor of *B* which also receives from *W*. Let the *REQ* from *W* be the first to arrive at both *B* and *E*, Figure 2(a). If nodes *B* and *E* forward the first *REQ* they receive and drop the others as in AODV, then multiple paths will be formed with *W* in them, Figure 2(b). However, using our technique, assuming that the timer of *E* runs out before that of *B* and that *E* broadcasts the message it received from *W*, then *B* will drop *W*'s packet from its buffer. The resulting paths are thus disjoint, Figure 2(c).

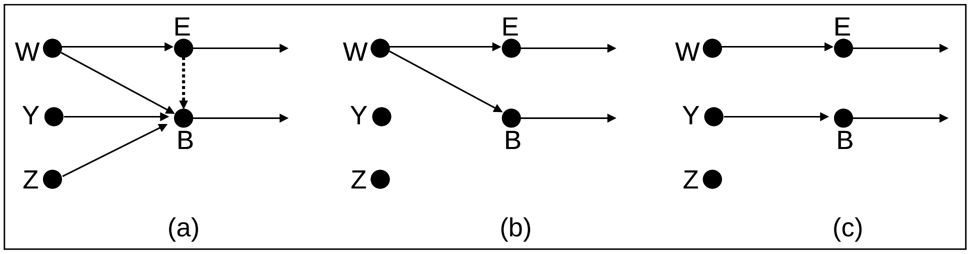


Figure 2: Example of node-disjoint routes.

The destination replies to every *REQ* copy it receives through a *different* neighbor. An intermediate node creates a routing table entry when it forwards the reply for the first time. Subsequently, it does not forward any further replies to prevent itself from being inserted in multiple routes. In order to detect malicious behavior by its neighbors, each node monitors replies going out of the neighbors. If a neighbor forwards a specific reply more than once, it is considered malicious and dropped from all the routes the node has. For example, let node *B* receive the *REP* packets for a given route creation procedure from two non-neighbor nodes *W* and *Y*. A correct node forwards only the first *REP*. However, if *B* is malicious, it may send the two replies to two different neighbors, say A and α respectively. Therefore, *B* succeeds in including itself in two "different routes". However, in LSR, this misbehavior can be detected by *W* and *Y* since they overhear *B*'s forwarded *REP*s. Then they evict all the routes through *B*.

5 Attacks and Countermeasures

In this section, we present three representative attacks that can be launched against a routing protocol and show how they can be detected in LSR with UNMASK. We also present a representative attack that can be launched against the data traffic and show how it can be detected using UNMASK.

5.1 ID Spoofing and Sybil Attacks

In this attack, an attacker presents one (ID spoofing) or more (Sybil attack) spoofed identities to the network [11]. Those identities could either be new fabricated identities or stolen identities from legitimate nodes. The Sybil attack can have many adverse impacts, such as, multipath routing [12] and collaborative protocols that use aggregation and voting [34].

Using UNMASK with LSR yields the following desirable properties to mitigate the ID spoofing and Sybil attacks:

(i) The first-hop neighbor list data structure prevents a node from spoofing the identity of a non-neighbor node. A node will not accept (forward) traffic from (to) a non-neighbor node. (ii) The one-hop source authenticated broadcasting prevents a node from generating traffic using spoofed identity of a neighbor node since each node must authenticate its generated traffic to the neighbors. (iii) Local monitoring detects a forwarding node when spoofing a neighbor's identity. As shown in Figure 1, if *A* receives a packet from *Y*, then *A* can not forward the packet claiming that it is being forwarded by one of its neighbors, say *W*. None of the guards of *W* over the link from *Y* to *W* overhear such a packet; also the guards of *A* over the link from *Y* to *A* accuse *A* of not forwarding the packet.

5.2 Wormhole Attack

In the wormhole attack [16],[17] a malicious node captures packets from one location in the network, and "tunnels" them to another malicious node at a distant point, which replays them locally. The tunnel can be established in many different ways, such as through an out-of-band hidden channel (e.g., a wired link), packet encapsulation, or high powered transmission. The tunnel creates the illusion that the two end-points are very close to each other, by making tunneled packets arrive either sooner or with a lesser number of hops compared to the packets sent over normal routes. This allows an attacker to subvert the correct operation of the routing protocol, by controlling numerous routes in the network. Once traffic is forced to flow through a node, it may launch denial of service against the data traffic. Notice that the wormhole attack can be launched without having access to any cryptographic keys or compromising any legitimate node.

UNMASK enables detection and isolation of malicious nodes launching wormhole attacks as follows:

Local monitoring detects the nodes involved in tunneling the route control packets and local response disables the tunnel from being established in the future by isolating the malicious nodes. Each guard saves the *SN*, the type, the source, the destination, the immediate sender, and the immediate receiver of every input packet to the monitored node. Consider the scenario in Figure 3. Two colluding nodes, M_1 and M_2, use an out-of-band channel or packet encapsulation to tunnel routing information between them. When M_1 receives the *REQ* initiated by *S*, it tunnels the *REQ* to M_2. Node M_2 has two choices for the previous hop — either to append the identity of M_1, or append the identity of one of M_2's neighbors, say *U*. In the first choice all the neighbors of M_2 reject the *REQ* because they all know, from the stored data structure of the two-hop neighbors, that M_1 is not a neighbor to M_2. In the second case, all the guards of the link from *U* to M_2 (*U*, *N*, and *L*) detect M_2 as fabricating the route request since they do not have the information for the corresponding packet from *U* in their watch buffer. In both cases M_2 is detected, and the guards increment the *MalC* of M_2. Similarly, when M_1 receives the *REP* tunneled from M_2 it has the same choices as M_2 and a similar scheme is used by the guards of the incoming link to M_1.

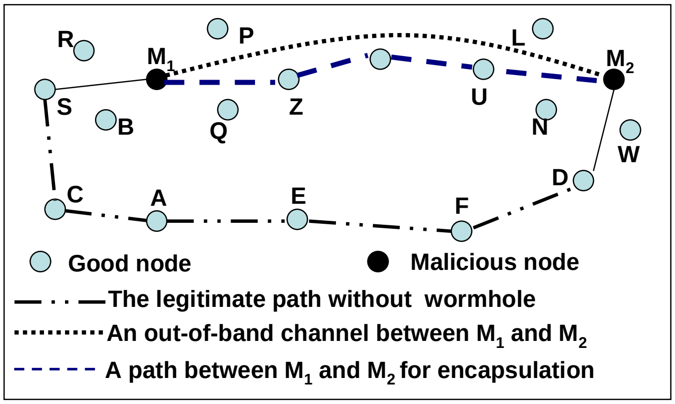

R
P
L
M1
M2
S
B
Q
Z
U
N
W
C
A
E
F
D
Good node
Malicious node
The legitimate path without wormhole
An out-of-band channel between M1 and M2
A path between M1 and M2 for encapsulation

Figure 3: A wormhole attack scenario

An alternate technique for attracting data traffic is through the *rushing attack* whereby a malicious node forwards the *REQ* packet without waiting. This is defeated in LSR since an intermediate node does not forward the first route request it receives (may be from a rushing malicious node), but rather, waits and collects copies of the *REQ* from different neighbors and randomly selects one of them to rebroadcast.

5.3 Selective Forwarding Attack

This is an example of an attack launched against the data traffic, where the adversary node selectively drops packets flowing through it. The attack can impact the end-to-end throughput in the network and if a reliable, continuous message stream is required, then this causes wastage of resources by inducing repeated retransmissions.

UNMASK enables the detection of selective forwarding as follows:

Information about the incoming data packet is stored in the watch buffer of the guard node. If the incoming packet stays in the watch buffer unmatched beyond a threshold period of time, the guard node increments the *MalC* value for the node being monitored. In the case of the selective forwarding attack, the packet which is dropped by the adversary node, will remain unmatched in the guard node's watch buffer. The guard node monitors a fraction of the data traffic, with the packet to be monitored being chosen randomly. This decision is independent of the decision of the adversary node to drop packets and therefore there is a vanishingly small probability that the set of packets dropped and the set of packets *not* monitored will exactly match over the time window over which the *MalC* value is aggregated. The adversary node will thus be detected when the *MalC* value crosses the threshold.

6 UNMASK Analysis

6.1 Coverage Analysis

In this section, we quantify the probability of missed detection and false detection of a generic attack as the network density increases and the detection confidence index varies. The results provide some interesting insights. For example, we are able to find the required network density *d* to detect *p*% of an attack under consideration for a given detection confidence index γ.

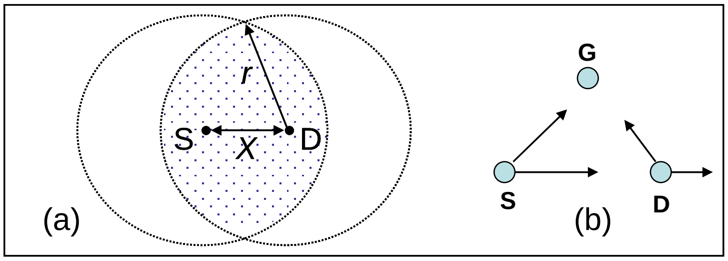


Figure 4: (a) The area where a node can guard the link between S and D; (b) Illustration for detection accuracy

Consider a homogeneous network where the nodes are uniformly distributed in the field. Consider any two randomly selected neighbor nodes, *S* and *D* (Figure 4(a)). Nodes *S* and *D* are separated by a distance *X,* and the communication range is *r*. *X* is a random variable with probability density function of $f_X(x) = 2x/r^2$ with range (0,*r*). This follows from the assumption of uniform distribution of the nodes.

The guard nodes for the link between *S* and *D* are those nodes that lie within the communication range of *S* and *D*, the shaded area in Figure 4(a). This area is given by $Area(x) = 2r^2\cos^{-1}\left(\frac{x}{2r}\right) - x\sqrt{r^2 - \frac{x^2}{4}}$. The minimum value of *Area(x), Area*$_{min}$, is when $x = r$.

$$E\left[Area(x)\right] = \int_0^r \left\{2r^2\cos^{-1}\left(\frac{x}{2r}\right) - x\sqrt{r^2 - \frac{x^2}{4}}\right\}\left(\frac{2x}{r^2}\right)dx \approx 1.84r^2$$

Therefore, the expected number of guards is $g = E[Area(x)]d = \lfloor 1.84r^2 d \rfloor$. The number of neighbors of a node is given by $N_B = \pi r^2 d$. Hence, $g \approx \lfloor 0.59 N_B \rfloor$ —— (I).

Now, as in [28] where IEEE 802.11 was analyzed, we assume that each packet collides on the channel with a constant and independent probability, P_C.

6.1.1 Analysis for Missed Detection

Following Figure 4(b), the four malicious actions may be missed due to different combinations of events. Drop are missed if there is a collision on the S→G link, fabrication for the D→G link, and modification and delay for both S→G and D→G links. If the attacker delays packets with probability P_{delay}, drops with probability P_{drop}, fabricates with probability P_{fab}, and modifies with probability P_{mod}, then, the probability of missed detection $P_M = (P_{drop} + P_{fab})\ P_C + (P_{mod} + P_{delay})P_C^2$. When plotting the probability of missed detection, we use equiprobable malicious actions (P_M = ¼(2P_C+2P_C^2)). Assume that μ packet attacks (fabrication, modify, drop, etc.) occur within a certain time window, *T*, with the different attacks being equiprobable. Also assume that a guard must detect at least β attacks to cause the *MalC* for a node to cross the threshold, and thus generate an alert and the increment to *MalC* is the same for each activity. Then, the alert probability at a guard is given by $P_{SG} = \sum_{i=\beta}^{\mu} \binom{\mu}{i} \left(1 - P_M\right)^i \left(P_M\right)^{\mu - i}$.

Thus, assuming independence of collision events among the different guards, the probability that at least γ of the guards generate an alert, i.e., the probability of detection is given by $p_{alert}(\geq \gamma) = \sum_{i=\gamma}^{g} \binom{g}{i} (P_{SG})^{i} (1 - P_{SG})^{g-i}$

Figure 5 shows the probability of detecting an attack (e.g. the wormhole) with μ = 7, β = 5, γ = 3, the number of compromised nodes M = 2, and P_C = 0.10 at N_B = 3. Thereafter, P_C is assumed to increase linearly with the number of neighbors (note that we do not use power control in the network). The number of guards is determined from N_B using Equation (I). Since the number of guards increases as the number of neighbors increases, the probability of detection increases since it becomes easier to receive the alarm from γ guards. However, the collision probability also increases with increasing node density, and thus the probability of detection starts to fall rapidly at a point. Beyond 24 neighbors, the collision is so high that the detection probability becomes zero. Figure 10 shows the probability of detection against γ with N_B=15 and similar parameter values as those of Figure 5.

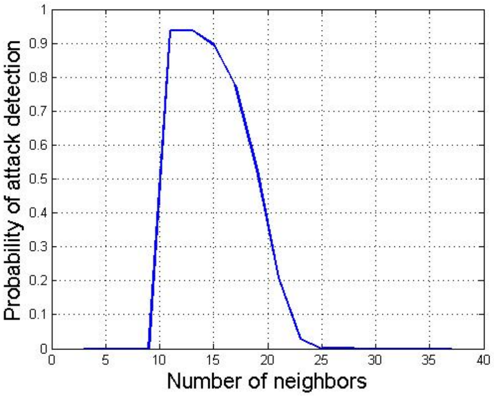


Figure 5: Probability of wormhole detection

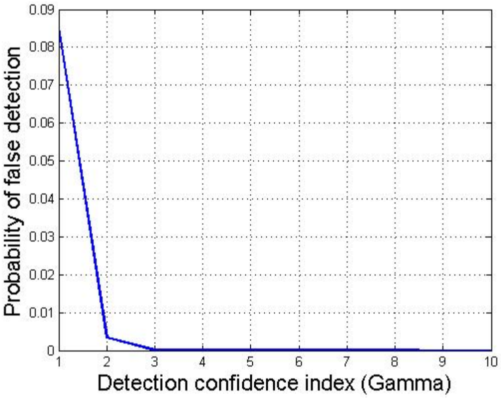


Figure 6: Probability of false alarm

6.1.2 Analysis for False Detection

Following Figure 4(b), a false alarm occurs due to falsely implicating a node for dropping, delaying, fabricating, or modifying packets. The false detection of each activity is caused by a different set of events – *drop* through no collision on the S→G link and either collision on the S→D link or no collision on the S→D link and collision on the D→G link; *fabrication* through collision on the S→G link and no collision on the S→D link and the D→G link. According to our model for analysis, a modified packet cannot give rise to false detection and a delay is not possible either since it will map to drop at the guard. The events for drop and fabrication are disjoint and therefore the individual probabilities are summed to give the combined probability of false alarm as $P_{FA} = 2P_C(1-P_C)^2 + P_C(1-P_C)$. Assume that S sends μ packets to D for forwarding within a certain time window, T. The probability that D is falsely accused is the probability that D is suspected of malicious actions for β or more packets. Thus, the probability of false alarm by a single guard is given by $P_{FA(\beta|\mu)} = \sum_{i=\beta}^{\mu} \binom{\mu}{i} (P_{FA})^{i} (1-P_{FA})^{\mu-i}$, and the probability that at least γ guards generate false alarms is given by $p_{FA}(\geq \gamma) = \sum_{i=\gamma}^{g} \binom{g}{i} \left(P_{FA(\beta|\mu)}\right)^{i} \left(1-P_{FA(\beta|\mu)}\right)^{g-i}$. Figure 6 shows the probability of false alarm as a function of γ with P_C = 0.05, β=4, μ=7, and N_B=10. As γ increases, the probability of false detection decreases since it becomes harder to reach consensus among all the γ guard nodes.

6.1.3 Analysis of Node Being Framed

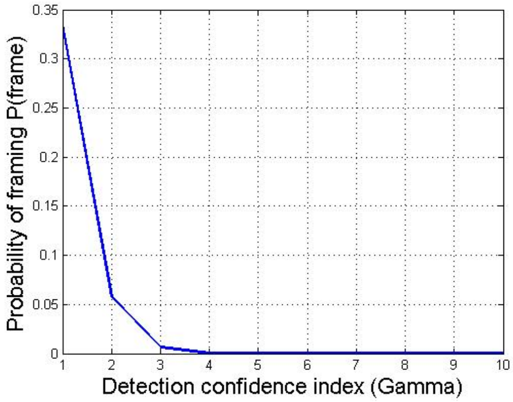


Figure 7: Probability of node framing

Let N be the total number of nodes in the network, M be the number of malicious nodes, P_m=M/N be the probability that a node gets compromised. Assuming that false detection is zero, then, the probability that a good node W is locally framed equals the probability that there are at least γ malicious nodes among W's neighbors, $P_{frame}(\gamma) = \sum_{i=\gamma}^{N_B} \binom{N_B}{i} P_m^i \left(1 - P_m\right)^{N_B - i}$.

Figure 7 shows *P(frame)* as a function of γ for $N = 100$, M=4, and $N_B = 10$. The probability of framing decreases exponentially with γ and goes to zero when γ>M.

6.1.4 Selection of the Detection Confidence Index (γ) Value

The value of γ is application-specific and may range between one and infinity. A small value for γ increases the chance of successful framing, while a large value of γ increases the harm a malicious node may cause before being locally detected and isolated. If we set γ to be infinity it means that a node only trusts itself in revoking a suspicious node, thus the local framing probability goes to zero. Any malicious node may be fully isolated as long as γ or more good-guards detect it. If the number of good guards is less than γ, then the node is only partially isolated from the network. Only the good guards that directly detect the malicious activity of the node isolate the malicious node. However, other neighbors of the malicious node continue to consider the malicious node as a legitimate node. The effect of partial or full isolation of malicious nodes is studied through the simulations (Figure 20, Figure 21, and Figure 22). In the simulations, the nodes are distributed randomly with a given density and the malicious nodes are also distributed randomly in the sensor field. The simulations include the case when the number of good guards of a node may fall below γ which negatively impacts the delivery ratio. However, the protocol as a whole does not suffer from a collapse of the delivery since this case is rare for a reasonable ratio of malicious nodes to the total number of nodes in the network. Moreover, based on our mathematical analysis presented here, we examine the effect of changing the value of γ in the network. Our simulation and analytical results indicate that a value of γ equal to infinity provides a good balance for many configurations, including the case where the volumes of traffic on the different outgoing links are statistically equal. Hence, γ can be looked upon as a design parameter in UNMASK to tune its performance according to the application needs.

6.2 Cost Analysis

The memory, computation, and bandwidth overhead of UNMASK are tolerable for resource constrained environments, such as sensor networks. For memory, each node needs to store a first and a second hop neighbor list, a commitment key for each first hop neighbor, its own commitment string, a watch buffer, and an alert buffer. The runtime state with fluctuating size is the watch buffer, whose size is higher if the guard is monitoring a malicious node that is delaying or dropping packets. The time for which the packet is kept in the watch buffer is relatively small, being determined by the MAC layer delay for acquiring the channel. From the experiments presented in the next section, we find that a watch buffer of size 50 is sufficient for all the experimental conditions. Each entry in the watch buffer is 14 bytes – 2 bytes each for the immediate source, the immediate destination, and the original source, and 8 bytes for the sequence number of the *REP* (*REQ*). The computation overhead is negligible since the operations for each message is lookup and addition or deletion in the small watch buffer. The bandwidth overhead is incurred only during initialization and when an adversary is detected. Assuming nodes are awake, the listening due to the role of a guard does not incur any bandwidth overhead.

7 Simulation Results: Control Attacks

We use the *ns-2* simulator [29] to simulate a data exchange protocol over LSR, individually without UnMask (the *baseline*) and with UnMask. We distribute the nodes randomly over a square area with a fixed average node density. Thus, the length of the square varies (80m to 204m) with the number of nodes (20-250). This random distribution may result in situations where the number of good guards of some nodes goes below γ, which negatively impact the simulation results. We first simulate the wormhole attack using out-of-band direct channels between the colluding nodes. The malicious nodes are randomly selected from the network nodes. After a wormhole is established, the malicious nodes at each end of the wormhole drop all the packets forwarded to them.

Each node acts as a source and generates data according to a Poisson process with rate μ. The destination is chosen at random and is changed using an exponential random distribution with rate ξ. A route is evicted if unused for $TOut_{Route}$ time. *Isolation latency* is defined as the time between when the node performs its first malicious action to the time by which *all* the neighbors of the node have isolated it.

The experimental parameters are given in Table 3. The results are averages over 30 runs. The malicious nodes are chosen at random such that they are more than 2 hops away from each other.

Table 3: Input parameter values

Parameter	Value	Parameter	Value
Tx Range (r)	30 m	γ	2-8
N_B	8	μ	100
$TOut_{Route}$	50 sec	M	0-10
τ, N_r	0.05 s, 5	β	5
Channel BW	40 kbps	ξ	5

Figure 8 shows the number of packets dropped as a function of simulation time for the 100-node setup with 2 and 4 colluding nodes. The attack is started 50 seconds after the start of the simulation. Since the numbers are vastly different in the baseline and with UnMask, they are shown on separate Y-axes (the left corresponding to the Baseline and the right corresponding to the UnMask case). In the baseline case, since wormholes are not detected and isolated, the cumulative number of packets dropped continues to increase steadily with time. But in UnMask, as wormholes are identified and isolated permanently, the cumulative number stabilizes. Note that the cumulative number of packets dropped grows for some time even after the wormhole is locally isolated. This is because the cached routes that contain the wormhole continue to be used until route timeout occurs.

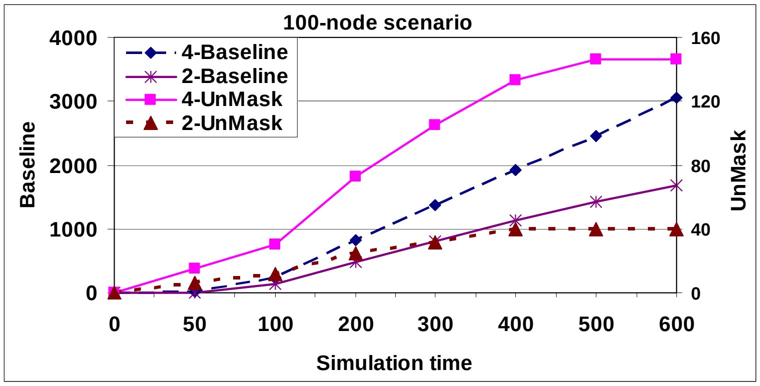


Figure 8: Cumulative no. of dropped packets

Figure 9 shows a snapshot, at simulation time of 2000 seconds, of the fraction of the total number of packets dropped to the total number of packets sent, and the fraction of the total number of routes that involve wormholes to the total number of routes established. This is shown for 0-4 compromised nodes for the baseline and with UNMASK. With 0 or 1 compromised node, there is no adverse effect on normal traffic since no wormhole is created. The relationship between the number of dropped packets and the number of malicious routes is not linear. This is because the route established through the wormhole is more heavily used by data sources due to the aggressive nature of the malicious nodes at the ends of the wormhole. If we track these output parameters over time, with UNMASK, they converge to zero as no more malicious routes are established or packets dropped, while with baseline case they would reach a steady state as a fixed percentage of traffic continues to be affected by the undetected wormholes.

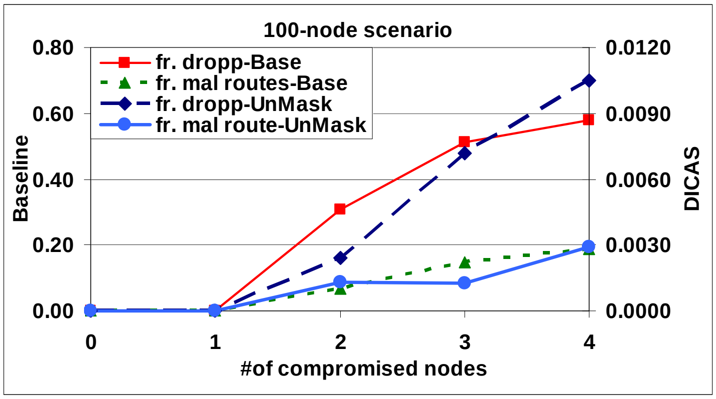


Figure 9: Fraction of dropped packets & malicious routes

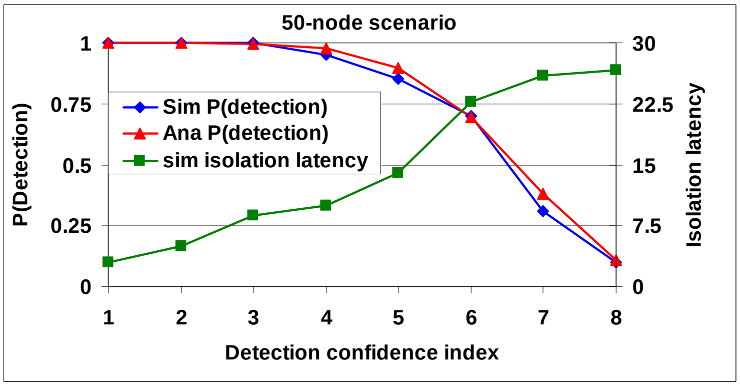


Figure 10: Detection probability & isolation latency

Figure 10 bears out the analytical result for the detection probability as γ is varied with N_B = 15 and M = 2. As γ increases, the detection probability goes down due to the need for alarm reporting by a larger number of guards, in the presence of collisions. Also the isolation latency goes up, although it is very small (less than 30 seconds) even at the right side of the plot.

Next, we simulate combined rushing and Sybil attacks over a network of 250 nodes deployed in a 300 m × 300 m field. We compare the average number of node-disjoint paths discovered per route request of an ideal search algorithm, AODVM [27], and LSR with UNMASK. In the ideal search, the topology of the entire network is known to the source that uses the shortest path first search algorithm. AODVM creates node-disjoint routes by having every node overhear neighboring nodes' *REP* packets and deciding to forward its own *REP* such that a neighbor is not included in two routes for a given source-destination pair. However, it does not consider any control attacks.

Figure 11 shows the average number of node-disjoint paths as a function of the number of hops in the shortest path between two nodes. The figure shows that, in a failure free environment, LSR and AODVM perform almost identically. In a malicious scenario (AODV malicious and UNMASK malicious scenarios), each of 10 malicious nodes launches rushing and Sybil attacks. When a malicious node receives a *REQ* packet, it rushes to broadcast N_r copies of the *REQ*, each with a different fake identity. Figure 11 shows that LSR with UNMASK is robust to the attack (LSR and LSR_mal plots overlap), while the average number of node-disjoint paths in AODVM is reduced by 22% (for distant source-destination pairs) to 32% (for closer pairs). Note that as the length of the path increases, the effect of the attacks in AODVM decreases. This is because even though the multiple routes appear to be disjoint at the attacker they may converge at some other intermediate node. These are then discarded by the source thereby ultimately foiling the attacker's goal.

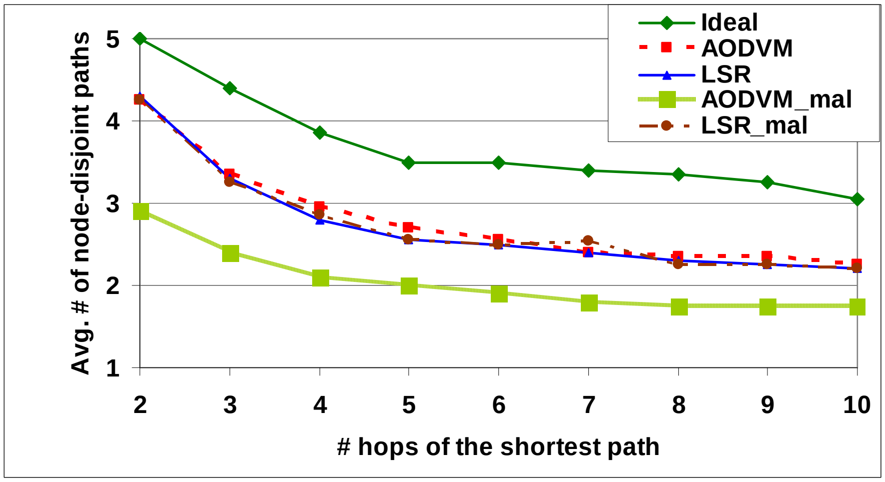


Figure 11: Average number of node-disjoint paths in ideal case, AODVM, and LSR

8 Simulation Results: Data Attacks

Adversary model: We are simulating the selective forwarding attack launched by a group of malicious nodes in two attack scenarios. In the first scenario, the malicious nodes collude and establish wormholes in the network. In the second scenario, the malicious nodes are independent and each node performs selective forwarding without any collusion or coordination with other malicious nodes. Unless otherwise mentioned, we use the wormhole adversary nodes. Each node selectively drops a fraction 0.6 of the traffic that passes through it.

Input metrics: *Fraction of data monitored* (f_{dat}) – each guard node randomly monitors a given fraction of the data packets. At other times, it can be asleep from the point of view of a guard's responsibility. *Increment to malicious counter* – This is the increment that a guard node does to the malicious counter for a given node for a single malicious action.

Output metrics: *Delivery ratio* – The fraction of the number of packets delivered to the destination by the number of packets sent out by a node averaged over all the nodes. *Watch buffer size* – This is the runtime count of the maximum size of the watch buffer being maintained at a guard, measured in number of entries. The maximum is taken over all the guards.

Simulation parameters: Here, we mention the parameter settings that are different from the experiments on control attacks. Unless explicitly varied as a control parameter in an experiment, the total number of nodes in the network $N = 100$, destination change rate $\lambda = 50$, $\gamma = 3$, *MalC* threshold beyond which a node is determined to be erroneous is 150, and the number of malicious wormhole nodes $M = 4$. The simulation time is 1500 seconds.

8.1 Algorithm for selection of *MalC* increment

An important design parameter in UNMASK is the increment to the malicious counter value upon detecting a malicious event. On the one hand, we want the increment to be large for higher detection probability, fast detection, and small watch buffer size. On the other hand, we want the increment to be small to reduce the percentage of false alarms. We conduct an experiment to design the malicious counter increment of a network with $f_{dat} = 0.4$ and number of wormhole nodes = 4. For the purpose of this experiment, we look at the increment for dropped messages.

Figure 12 shows that the percentage of false alarms increases as the *MalC* increment increases. With higher *MalC* increment, the chance that natural errors, such as collisions, cause the *MalC* to reach the threshold becomes higher, which results in an overall increase in the percentage of false alarms. The figure also shows that the detection percentage increases as the *MalC* increment value increases to a point (increment = 7) after which it remains approximately constant. As the size of the increment increases, a smaller number of events causes the *MalC* threshold to be reached which enhances the opportunity of detecting malicious nodes, even those that are involved in a small number of malicious events. The delivery ratio also increases with increasing *MalC* increment value to a point (*MalC* increment = 7) after which it remains approximately constant. Faster detection results in fewer number of dropped data packets. However, the rate slows down beyond a point since any additional increase does not substantially accelerate the process.

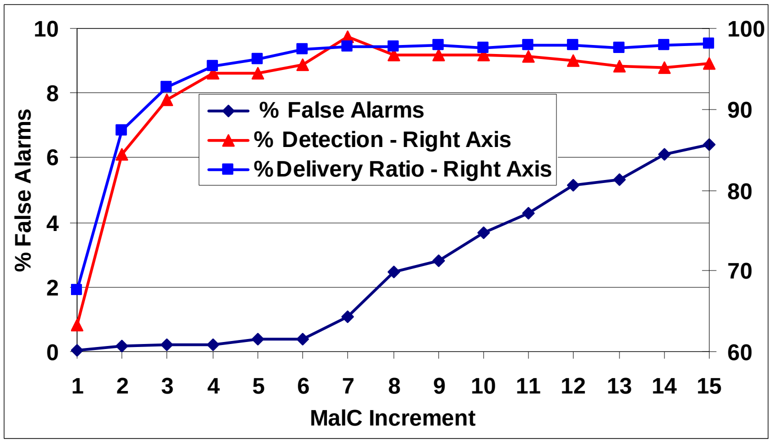


Figure 12: Effect of *MalC* increment

For the rest of the experiments in the section, for each given f_{dat}, we choose the increment as the lower of the two points – the point where the percentage detection reaches its maxima and the point where the knee of the false detection curve lies. This gives us a reasonable combination of low false alarm rate and high detection rate. The values of *MalC* increment used for the rest of the experiments are summarized in Table 4.

Table 4: MalC increment per malicious activity used for the experiments

Fraction of data monitored	*MalC* Increment
0.2	11
0.4	8
0.6	5
0.8	2
1.0	1

8.2 Effect of Fraction of Data Monitored (f_{dat})

The amount of data traffic is typically several orders of magnitude larger than the amount of control traffic. It is not reasonable for a guard node to monitor all the data traffic in its monitored links. Therefore a reasonable optimization, as proposed in Section 3.4, is to monitor only a fraction of the data traffic. In this set of experiments, our goal is to investigate the effect of this optimization on the output metrics.

Figure 13 shows the variations of delivery ratio as we vary f_{dat} with four wormhole malicious nodes. The *MalC* increment for each f_{dat} is designed as shown in Section 8.1 with an inverse relation to the f_{dat}. The selection of the *MalC* increment value according to the algorithm keeps the delivery ratio almost stable and above 95%, irrespective of f_{dat}.

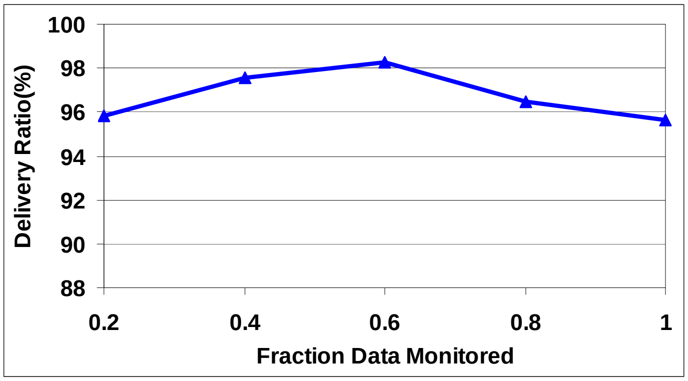


Figure 13: Effect of fraction of data monitored on delivery ratio

Figure 14 shows that the percentage of false alarms decreases as f_{dat} increases. More available data makes it easier to distinguish a good node from a malicious node. The higher the value of f_{dat}, the lower is the increment to the malicious counter and thus the smaller the chance of reaching the malicious counter threshold by natural errors only. These two factors help reduce the probability of false alarms with increasing f_{dat}. Figure 14 also shows the variations of detection percentage as we vary f_{dat}. By selecting the appropriate *MalC* increment value, we manage to keep the detection percentage almost stable and above 95% irrespective of f_{dat}. As f_{dat} increases, *MalC* increment decreases. This causes the *MalC* threshold to be reached slower at a guard node, which results in increasing the isolation latency of the malicious nodes, Figure 15. Also the higher f_{dat} lays it open to the possibility of some packets being missed due to natural collisions and thereby preventing the increment to the malicious counter and therefore, reaching the threshold. Note however, that the delivery ratio is largely unaffected (Figure 13) since a malicious node may still not be completely isolated by all its neighbors. However, it does not have the opportunity for too much damage since most of its neighbors have already isolated it and when new routes are created, the malicious node is excluded. As the value of f_{dat} increases, the size of the watch buffer expectedly increases, Figure 15. This increases the overhead of local monitoring since a larger memory has to be maintained and searched in.

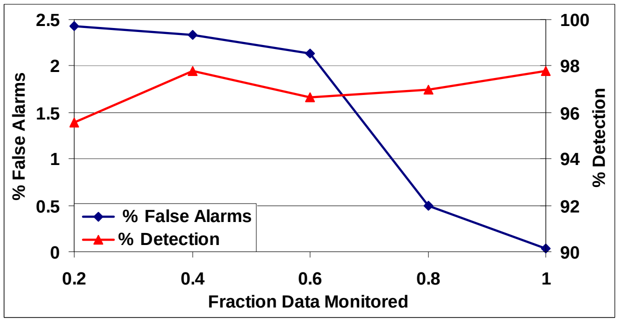


Figure 14: Percentage detection and % false alarms

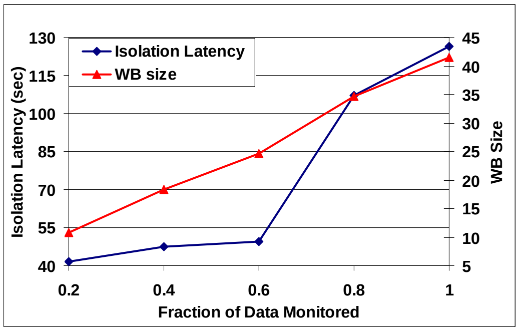


Figure 15: Isolation latency and Watch buffer size

8.3 Effect of Number of Malicious Nodes

Figure 16 shows the effect of increasing the number of malicious nodes when launching two different scenarios of attacks – the perfectly colluding wormhole nodes and the independent adversary nodes. Note that in both scenarios, the delivery ratio falls almost linearly as we increase the number of malicious nodes from 2 to 6. This is due to the packets dropped before the malicious nodes are detected and isolated. As the number of malicious nodes increases, this initial drop increases and thus the delivery ratio decreases. A second-order effect for the decrease in the delivery ratio is the decrease in the number of available guards making it more difficult to obtain agreement from γ guard nodes. However, the delivery ratio is always above 92% for the wormhole scenario and above 96% for the independent scenario. Note also that the delivery ratio in the independent scenario is higher than that in the wormhole scenario. This is due to the aggressive nature of the wormhole which attracts traffic from many nodes through the malicious nodes and increases the initial traffic dropped before the malicious nodes get isolated.

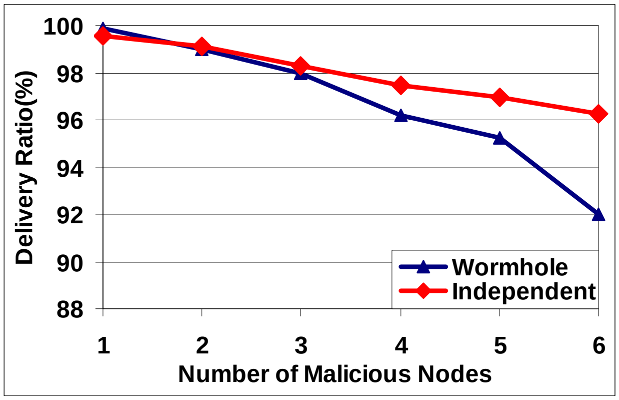


Figure 16: Delivery ratio as a function of malicious nodes

Figure 17 shows the percentage of false alarms and the percentage of detection as a function of the number of malicious nodes. The percentages of false alarms increases as the number of malicious nodes increases because not all guard nodes come to the decision to isolate a malicious node at the same time. Therefore a given guard node may suspect another guard node when the latter isolates a malicious node but the former still has not. For example, a guard node G_1 detects a malicious node M earlier than the other guard nodes for the link to M. G_1 subsequently drops all the traffic forwarded to M and is therefore suspected by other guard nodes for M. This problem can be solved by having an authenticated one-hop broadcast whenever a guard node performs a local detection. The detection percentage falls almost linearly as we increase the number of colluding malicious nodes from 2 to 6 due to the decrease in the number of available guards. However, the detection percentage is always above 88% in all our experiments.

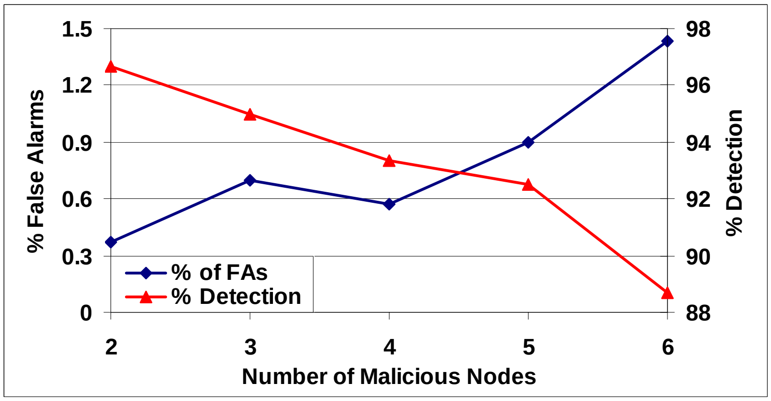


Figure 17: False alarms and detection as a function of number of malicious nodes

Figure 18 shows the isolation latency and the watch buffer size as a function of the number of malicious nodes. As the number of malicious nodes increases, the isolation latency slightly increases. This is due to the fact that an individual malicious node has less opportunity to do harm, which delays its detection and thus increases the average isolation latency. As we increase the number of malicious nodes, the watch buffer size increases since a larger number of packets stays longer in the watch buffer waiting to be matched since these packets are eventually dropped by the malicious nodes.

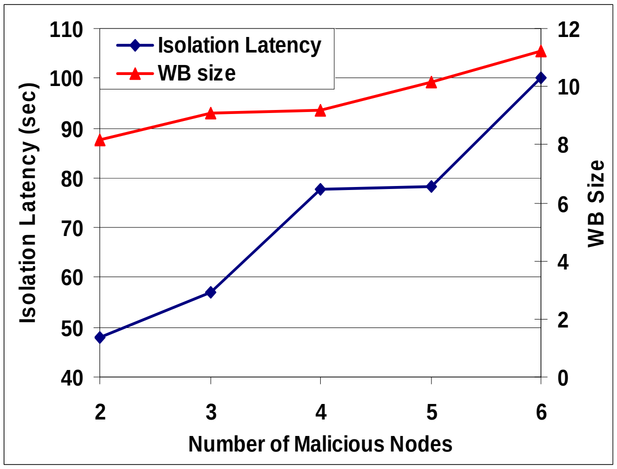


Figure 18: Isolation latency and watch buffer size as a function of number malicious nodes

8.4 Effect of varying the Detection Confidence Index γ

Figure 19 shows the percentage of framing with various values of *γ*. As the number of malicious nodes increases, the chances of getting *γ* malicious nodes framing a good node increases and thus the framing percentage increases. As we increase *γ*, the percentage of framing decreases since it becomes more difficult to get *γ* malicious nodes to frame a good node. When the value of *γ* is greater or equal to 7, the probability of framing goes to zero since no node has more than 7 neighbors in this simulation setup, therefore, it is impossible for framing to occur. As *γ* increases however, as seen from Figure 20, Figure 21, and Figure 22, the damage done to the network before a malicious node is detected and isolated, increases. Note that here *γ* =7 is equivalent to *γ*=infinity since no node has more than 7 neighbors in the simulation setup, therefore, these two values represent the same results.

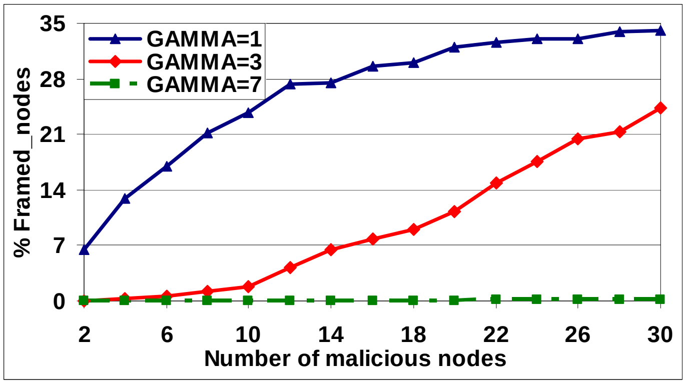


Figure 19: Percentage of framing

Figure 20 shows the percentage of false isolation as a function of *γ*. As *γ* increases the false isolation decreases since it becomes more and more unlikely to get *γ* nodes in a neighborhood that will mistakenly accuse a good node. As the number of malicious nodes increases the false isolation increases for the same reasoning as in Figure 19.

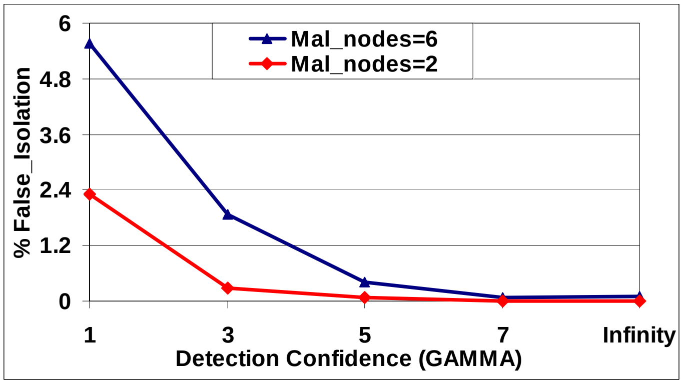


Figure 20: Percentage of false isolation

Figure 21 shows that the percentage of malicious routes (i.e., routes that include malicious nodes) increases with γ. As γ increases, the detection and isolation of nodes decreases and takes a longer time which gives the malicious nodes a greater chance to establish more malicious routes. Moreover, as the number of malicious nodes increases, the amount of damage (here, the number of malicious routes) increases.

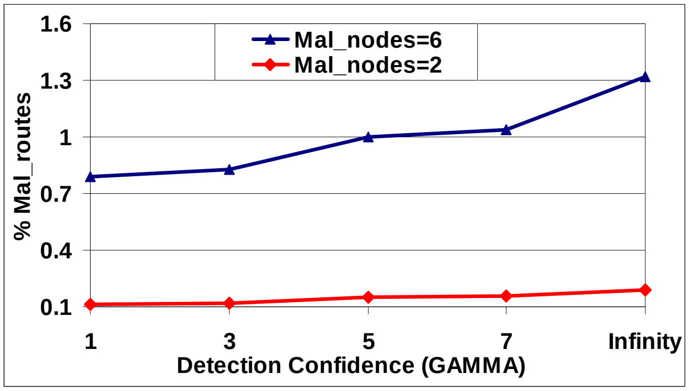


Figure 21: Percentage of malicious routes

Figure 22 shows the percentage of malicious nodes isolated at the end of the simulation time for three different values of γ. The isolation percentage falls almost linearly as we increase the number of colluding malicious nodes from 2 to 6 due to the decrease in the number of available guards. Note that as γ increases, the percentage of malicious nodes isolated decreases slightly due the requirement of higher number of guards to agree on the detection. However, the percentage of malicious nodes isolated is above 90% for 6 malicious nodes with infinite γ.

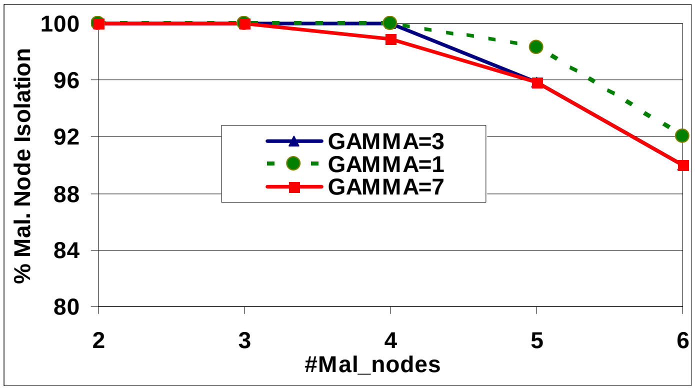


Figure 22: Percentage of malicious node isolation

The observation from these experiments is that an infinite value of γ appears to be a desirable operating region. We find that it eliminates framing and minimizes the percentage of false isolation. On the other hand, it only slightly increases the percentage of malicious routes and slightly decreases the percentage of malicious nodes isolated. Further, these values are close to the case when γ is small. This is because the guards of a node over a *certain link* are likely to see the same view of the node and therefore, they are likely to reach the same conclusion about the monitored node whether individually or through the reports of other guards. This reduces the importance of having guards inform each other of their view about the monitored node which results in little change when we increase the value of γ to infinity. However, the effect of the parameter will depend on the specifics of the network configuration and the traffic pattern. For example, here the traffic flows on all the links are statistically identical.

8.5 Energy Overhead

Energy overhead of monitoring involves– (i) the energy spent by the CPU running the specific details of the monitoring algorithm such as searching the buffers, reading and writing in the serial flash, (ii) the energy spent in sending/receiving packets related to monitoring such as neighbor discovery and malicious node detection announcements, (iii) and the energy spent in idle listening. The last ingredient depends on whether the network is implementing sleeping and on which sleeping technique is being used. For the detailed mathematical analysis of energy overhead in both cases we refer the reader to our previous work [51].

We implemented our detection algorithm on a testbed consisting of Crossbow Mica2 motes [50]. We use indirect measurements of energy consumption, namely, the time the CPU spends in the algorithm, the number of flash memory writes, and the time a node needs to be on just for monitoring purposes, which includes both the receive and the transmit time. Then we calculate the energy consumption using these measured parameters and the Mica2 data sheet values for the current draw [50]: CPU active = 8 mA, idle 3.3mA, sleep 8μA, Serial flash write 15mA, serial flash read = 4 mA, serial flash sleep 2μA, Radio Rx 10mA, Tx (max power) = 27mA. The watch buffer is maintained in the serial flash. The experiments are conducted on 50 nodes with 4 malicious nodes, $\gamma = 3$, $\mu = 10$, and $\lambda = 50$. We use for each node two Alkaline Long-life AA batteries (1.225 average voltages during discharge, each provide a total of 9360 joules). The experiment time is 60 minutes.

Figure 23 shows the results of the experiment conducted to measure the computational energy overhead. The figure shows the benefit, in terms of computational energy overhead, of monitoring only a small fraction of data. For this experiment, we implement the algorithm for storing packets in the watch buffer and searching in it through a linear search. The algorithm takes the size of the watch buffer as input. For the experiment, the maximum size of the watch buffer over all the guard nodes from the simulations is used. The algorithm is executed to search for a random number between 0 and 0.2 million. Since the size of the watch buffer is much smaller, most of the searches are unsuccessful mimicking a guard node overseeing a malicious node which is dropping packets. Since unsuccessful searches take longer than successful ones, this is another cause for overestimating the execution time. Since a smaller fraction of the data monitored results in smaller watch buffer sizes and fewer number of searches, the overhead with all the data being monitored is about 18 times the overhead with only a fraction 0.2 of the data being monitored.

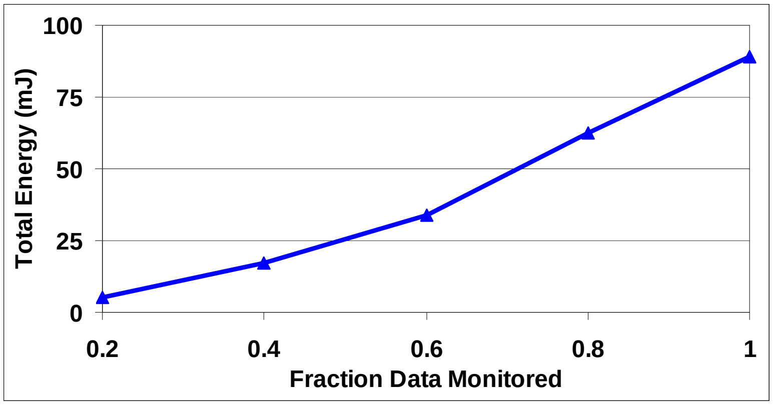
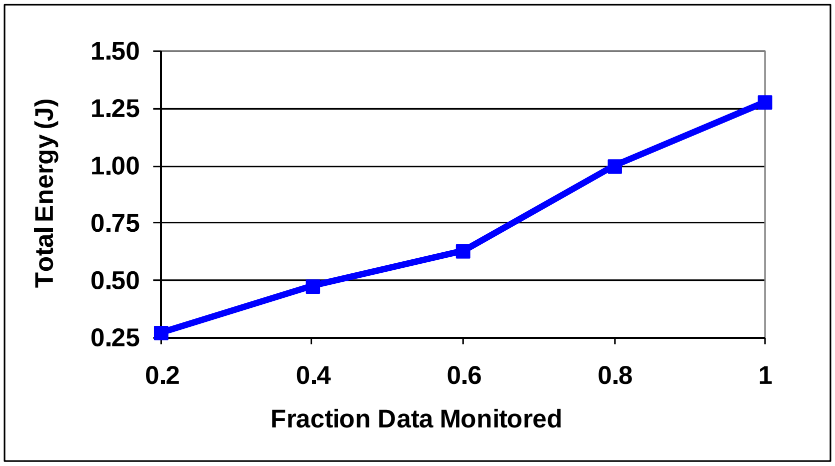


Figure 23: Computational energy consumed per node

Figure 24 shows the results of the experiment conducted to measure the total monitoring energy overhead, i.e., the sum of the factors (i), (ii), and (iii) mentioned earlier in this sub-section. Expectedly, Figure 24 shows that the total energy overhead increases as the fraction of data monitored increases. Note that the worst case total energy overhead (when the fraction of data monitored = 1.0) over a 1-hour period is less than 1.5 Joule which represents only 0.008 % of the total energy that the AA batteries can provide. This provides strong evidence that our protocol is parsimonious in its energy overhead and is, therefore, suitable for energy-constrained sensor networks. Moreover, note that the computational energy overhead (Figure 23), when f_{dat} = 1.0, is less than 7% of the total energy overhead.



9 Figure 24: The total overhead energy per-node due to monitoring over the experiment time (1 hour)Conclusion

We have presented a distributed protocol, called UnMask, for detection, diagnosis, and isolation of nodes launching control attacks, such as, wormhole, Sybil, rushing, sinkhole, and replay attacks. UnMask uses local monitoring to detect control and data traffic misbehavior, and local response to diagnose and isolate the suspect nodes. We analyze the security guarantees of UnMask and show its ability to handle attacks through a representative set of these attacks. We present a coverage analysis and find the probability of false alarm and missed detection. On top of UnMask, we build a secure lightweight routing protocol, called Lsr, which also supports node-disjoint path discovery.

We note that although designed for static networks, UnMask can potentially be extended to mobile networks. In mobile networks the neighborhood changes and therefore the neighbor discovery is required to be executed during the lifetime of the network. Therefore, the neighbor discovery protocol presented here cannot be secure for mobile networks. Note that incremental deployment of nodes is equivalent to a node moving to the new position and the situation can be handled similarly. As future work we are investigating secure neighbor discovery protocols appropriate for resource-constrained mobile networks.

10 References

[1] M. G. Zapata, Secure ad-hoc on-demand distance vector (SAODV) routing, IETF MANET Mailing List, October 8, 2001.

[2] Y.-C. Hu, D. B. Johnson, and A. Perrig, SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks, WMCSA 2002, pp. 3-13.

[3] Y.-C. Hu, A. Perrig, and D. B. Johnson, Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks, MobiCOM 2002, pp. 12-23.

[4] P. Papadimitratos and Z. Haas, Secure routing for mobile ad hoc networks, SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002), January 2002.

[5] C. Intanagonwiwat, R. Govindan, and D. Estrin, Directed diffusion: A scalable and robust communication paradigm for sensor networks, MobiCOM 2000, pp. 56-67.

[6] B. Karp and H. T. Kung, GPSR: greedy perimeter stateless routing for wireless networks, MobiCOM 2000, pp. 243-254.

[7] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, Mobile Computing and Communications Review, vol. 4, no. 5, October 2001, pp. 11-25.

[8] F. Ye, A. Chen, S. Lu, and L. Zhang, A scalable solution to minimum cost forwarding in large sensor networks, ICCCN 2001, pp. 304-309.

[9] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, Energy efficient communication protocol for wireless micro sensor networks, HICSS 2000, pp. 3005-3014.

[10] D. Braginsky and D. Estrin, Rumor routing algorithm for sensor networks, WSNA 2002, pp. 22-31.

[11] J. Newsome, E. Shi, D. Song, and A. Perrig, The Sybil attack in Sensor Networks: Analysis & Defenses, IPSN 2004, pp. 259-268.

[12] K. Ishida, Y. Kakuda, and T. Kikuno, A routing protocol for finding two node-disjoint paths in computer networks, ICNP 1992, pp. 340 347.

[13] Y. Xu, J. Heidemann, and D. Estrin, Geography-informed energy conservation for ad hoc routing, MobiCOM 2001, pp. 70-84.

[14] C. Karlof and D. Wagner, Secure Routing in Sensor Networks: Attacks and Countermeasures, SNPA 2003, pp. 113-127.

[15] S. Marti, T. J. Giuli, K. Lai, and M. Baker, Mitigating routing misbehavior in mobile ad hoc networks, MobiCOM 2000, pp. 255-265.

[16] Y. C. Hu, A. Perrig, and D.B. Johnson, Packet leashes: a defense against wormhole attacks in wireless networks, IEEE INFOCOM 2003, pp. 1976- 1986.

[17] L. Hu and D. Evans, Using Directional Antennas to Prevent Wormhole attacks, NDSS 2004, pp. 131-141.

[18] Y. C. Hu, A. Perrig, and D. Johnson, Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols, WiSe 2003, pp.30-40.

[19] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. Belding-Royer, A Secure Routing Protocol for Ad hoc Networks, ICNP 2002, pp. 78-87.

[20] S. Lindsey and C. Raghavendra, PEGASIS: power-efficient gathering in sensor information systems, IEEE Aerospace Conference 2002, vol. 3, 1125 - 1130.

[21] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, and D.E. Culler, SPINS: Security Protocols for Sensor Networks, Wireless Networks 2002., vol. 8, pp. 521-534

[22] D. Liu and P. Ning, Establishing Pair-wise Keys in Distributed Sensor Networks, CCS 2003, pp. 52-61.

[23] C. E. Perkins and E. M. Royer, Ad-Hoc On-Demand Distance Vector Routing, in Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99) 1999, pp. 90-100.

[24] P. Papadimitratos and Z.J. Haas, Secure Message Transmission in Mobile Ad Hoc Networks, WiSe 2003, pp.41-50.

[25] S.J. Lee and M. Gerla, Split Multipath Routing with Maximally Disjoint Paths in Ad Hoc Networks, ICC 2001, pp. 3201-3205.

[26] A. Nasipuri, R. Castaneda, and S.R. Das, Performance of Multipath Routing for On-demand protocols in Mobile Ad Hoc Networks, ACM Mobile Networks and Applications (MONET) 2001, 6(4): pp. 339-349.

[27] Z. Ye, S. V. Krishnamurthy, S. K. Tripathi, A Framework for Reliable Routing in Mobile Ad Hoc Networks, IEEE INFOCOM 2003, vol.1, pp. 270-280.

[28] G. Bianchi, Performance analysis of the IEEE 802.11 Distributed Coordination Function, IEEE Journal on Selected Areas in Communications, 2000, 18(3): pp. 535-547.

[29] The Network Simulator ns-2, At: www.isi.edu/nsnam/ns/

[30] A. A. Pirzada and C. McDonald, Establishing Trust In Pure Ad-hoc Networks, Proceedings of 27th Australasian Computer Science Conference (ACSC'04), pp. 47-54.

[31] S. Buchegger, J.-Y. Le Boudec, Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes - Fairness In Distributed Ad-hoc NeTworks, in MobiHoc 2002, pp. 80-91.

[32] Y. Huang and W. Lee, A Cooperative Intrusion Detection System for Ad Hoc Networks, SASN 2003, pp. 135-147.

[33] B. Schneier, Applied Cryptography, 2nd edition, Prentice Hall, 1996.

[34] M. Krasniewski, P. Varadharajan, B. Rabeler, S. Bagchi, Y. C. Hu, TIBFIT: Trust Index Based Fault Tolerance for Arbitrary Data Faults in Sensor Networks, DSN 2005, pp. 672 - 681.

[35] Secure Hash Standard, Federal Information Processing Standards Publication 180-1, April 1995.

[36] I. Khalil, S. Bagchi, and N. B. Shroff, LITEWORP: A Lightweight Countermeasure for the Wormhole Attack in Multihop Wireless Networks, DSN 2005, pp. 612-621.

[37] I. Khalil, S. Bagchi, and C. Nita-Rotaru, DICAS: Detection, Diagnosis and Isolation of Control Attacks in Sensor Networks, SecureComm 2005.

[38] Q. Zhang, P. Wang, D. S. Reeves, and P. Ning, Defending against Sybil Attacks in Sensor Networks, SDCS 2005, pp. 185-191.

[39] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru and H. Rubens, On the Survivability of Routing Protocols in Ad Hoc Wireless Networks, SecureComm 2005.

[40] C. Basile, Z. Kalbarczyk, and R. K. Iyer, Neutralization of Errors and Attacks in Wireless Ad Hoc Networks, DSN 2005, pp. 518-527.

[41] L. Lazos and R. Poovendran, SeRLoc: Robust localization for wireless sensor networks, in the ACM Transactions on Sensor Networks (TOSN), 2005, Volume 1 , Issue 1, pp. 73-100.

[42] R. Poovendran and L. Lazos, A Graph Theoretic Framework for Preventing the Wormhole Attack in Wireless Ad Hoc Networks, in ACM Journal on Wireless Networks (WINET), Volume 13, Number 1, 2007, pp. 27-59.

[43] B. Carbunar, I. Ioannidis and C. Nita-Rotaru, JANUS: Towards Robust and Malicious Resilient Routing in Hybrid Wireless Networks, WiSe 2004, pp. 11-20.

[44] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru and H. Rubens, On the Survivability of Routing Protocols in Ad Hoc Wireless Networks, in SecureComm 2005.

[45] L. Lamport, Password authentication with insecure communication, in Communications of the ACM, 24(11), pp. 770-772, November 1981.

[46] R. Hauser, T. Przygienda, and G. Tsudik, Reducing the Cost of Security in Link-State Routing, in Internet Society Symposium on Network and Distributed Systems Security, 1997.

[47] Y. Hu, A. Perrig, and D. Johnson, Efficient Security Mechanisms for Routing Protocols, in Proceedings of Network and Distributed Systems Security, 2003.

[48] S. Capkun, L. Buttyán, and J.-H. Hubaux, SECTOR: Secure Tracking of Node Encounters in Multi-hop Wireless Networks, in Proceedings of the 1st workshop on security of ad hoc and sensor networks, pp. 21-32, 2003.

[49] R. Molva, G. Tsudik, and D. Westhoff (Eds.), Statistical Wormhole Detection in Sensor Networks, ESAS 2005, LNCS 3813, pp. 128–141, 2005.

[50] http://www.xbow.com/products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf

[51] I. Khalil, S. Bagchi, and N. B. Shroff, “SLAM: Sleep-Wake Aware Local Monitoring in Sensor Networks,” in the 37th IEEE Dependable Systems and Networks Conference (DSN’07), p.p. 565-574, 2007.

Issa Khalil received the B.Sc. degree in Computer Engineering from Jordan University of Science and Technology (JUST), Jordan, in 1994, the MS degree in computer engineering from JUST in 1996, and the PhD degree in computer engineering from Purdue University, USA, 2006. Immediately thereafter he worked as a post-doctoral research associate at the Dependable Computing Systems Lab of Purdue University until he joined the College of Information Technology (CIT) of the United Arab Emirates University (UAEU) as an assistance professor in 2007. His research interests span the areas of wireless and wireline communication networks. He is especially interested in security, routing, and performance of wireless Sensor, Ad Hoc and Mesh networks. He has worked as an instructor and a director of the computer and communication center of Alquds Open University, West Bank, for more than 6 years.

Saurabh Bagchi joined the department of Electrical and Computer Engineering at Purdue University in West Lafayette, Indiana as an Assistant Professor in August 2002. Before that, he did his Ph.D. from the Computer Science department of the University of Illinois at Urbana-Champaign with Prof. Ravishankar Iyer at the Coordinated Science Laboratory. His Ph.D. dissertation was on error detection protocols in distributed systems and was implemented in a fault-tolerant middleware system called Chameleon.

Ness B. Shroff Ness B. Shroff received his Ph.D. degree from Columbia University, NY in 1994 and joined Purdue university immediately thereafter as an Assistant Professor. At Purdue, he became Professor of the school of Electrical and Computer Engineering in 2003 and director of CWSA in 2004, a university-wide center on wireless systems and applications. In July 2007, he joined The Ohio State University as the Ohio Eminent Scholar of Networking and Communications, and chaired Professor of ECE and CSE. His research interests span the areas of wireless and wireline communication networks. He is especially interested in fundamental problems in the design, performance, pricing, and security of these networks. Dr. Shroff is an editor for IEEE/ACM Trans. on Networking and the Computer Networks Journal, and past editor of IEEE Communications Letters. He has served on the technical and executive committees of several major conferences and workshops. He was the technical program co-chair of IEEE INFOCOM'03, the premier conference in communication networking. He was also the conference chair of the 14th Annual IEEE Computer Communications Workshop (CCW'99), the program co-chair for the symposium on high-speed networks, Globecom 2001, and the panel co- chair for ACM Mobicom'02. Dr. Shroff was also a co-organizer of the NSF workshop on Fundamental Research in Networking, held in Arlie House Virginia, in 2003. In 2008, he will serve as the technical program co-chair of ACM Mobihoc 2008. Dr. Shroff is a fellow of the IEEE. He received the IEEE INFOCOM 2006 best paper award, the IEEE IWQoS 2006 best student paper award, the 2005 best paper of the year award for the Journal of Communications and Networking, the 2003 best paper of the year award for Computer Networks, and the NSF CAREER award in 1996 (his INFOCOM 2005 paper was also selected as one of two runner-up papers for the best paper award).

Cristina Nita-Rotaru is an Assistant Professor in the Computer Science department of the Purdue University. She is the director of the Dependable and Secure Distributed Systems Laboratory. She received the BS and MS degrees in Computer Science from Politechnica University of Bucharest, Romania, in 1995 and 1996, and a PhD degree in Computer Science from The Johns Hopkins University in 2003. She served on the technical program committee of numerous conferences such as INFOCOM, ICDCS, ICNP, MOBIHOC. She received the National Science Foundation CAREER award in 2006. Her research interests include secure distributed systems, network security protocols in wired and wireless networks. She is a member of the ACM and IEEE Computer Society.