DICAS: Detection, Diagnosis and Isolation of Control Attacks in Sensor Networks

Issa Khalil, Saurabh Bagchi, Cristina Nina-Rotaru
Dependable Computing Systems Lab (DCSL)
School of Electrical & Computer Engineering, Purdue University
465 Northwestern Avenue, West Lafayette, IN 47907.
Email: {ikhalil, sbagchi, crisn}@purdue.edu Ph. 765-494-3362

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, Sybil attacks, rushing attacks, etc). In this paper we propose a lightweight protocol called DICAS that mitigates these attacks by detecting, diagnosing, and isolating the malicious nodes. DICAS uses as a fundamental building block the ability of a node to oversee its neighboring nodes' communication. On top of DICAS, *we build a secure routing protocol, LSR, which in addition supports multiple node-disjoint paths. We analyze the security guarantees of DICAS and use ns-2 simulations to show its effectiveness against three representative attacks. Overhead analysis is conducted to prove the lightweight nature of DICAS.*

Keywords: sensor network security, neighbor monitoring, secure routing, node-disjoint paths, control 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 them vulnerable to a wide range of security attacks. Some of the most devastating attacks target the control traffic or data traffic in wireless networks. Typical examples of control traffic are routing, monitoring the liveness of nodes, topology discovery, and distributed location determination. Control traffic attacks include the (i) wormhole attack ([20],[21]), (ii) the rushing attack [22], (iii) the Sybil attack [13], (iv) the sinkhole attack [18], and (v) the HELLO flood attack [18]. Attacks against data traffic include (vi) blackhole and (vii) selective forwarding [18] in which a malicious node drops entirely or selectively data passing through it. Control attacks are especially dangerous because they can be used to subvert the functionality of the routing protocol and create opportunity for a malicious node to launch a data traffic attack such as dropping all or a selective subset of data packets. Coping with control attacks in sensor networks is more challenging than in ad hoc wireless and wired networks due to the resource constrained environment.

In this paper we present a lightweight protocol called DICAS, which mitigates control traffic attacks in sensor networks. DICAS not only detects the occurrence of an attack, but also diagnoses the malicious nodes involved in the attack and removes their capability of launching future attacks by isolating them from the network. The detection and isolation mechanisms are executed locally, incurring only a small overhead. DICAS is suited to the low cost point of sensor networks since it does not require any specialized hardware (such as directional antennas [21] or GPS) nor does it require time synchronization among the nodes [20]. The approach that DICAS uses to achieve its security goals exploits a well-known technique whereby nodes oversee part of the traffic going in and out of their neighbors [19], [30], [35], [36]. Our novelty lies in presenting the technique as a standalone module – *local monitoring* – and analyzing its capabilities and limitations. We systematically lay out the fundamental structures and the state to be maintained at each node for mitigating five representative attacks – modifying routing traffic, Sybil, wormhole, sinkhole, and rushing attacks. Independent of the detection mechanism, we propose a strategy to isolate the malicious nodes locally in a distributed manner.

We use DICAS to build a lightweight secure routing protocol called LSR that withstands known attacks against the routing infrastructure and supports secure node-disjoint route discovery. We provide a security analysis of LSR using DICAS for five representative attacks. We analyze the detection coverage and the probability of false detection of DICAS. Also, we evaluate the memory, communication, and computation overhead of DICAS. Finally, we simulate the wormhole attack in *ns-2* and show its effect on the network performance with and without DICAS. The results show that DICAS 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 negligible time after the 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 DICAS. The results show that LSR using DICAS is resilient to the combined attack and that the average number of node-disjoint routes discovered is not reduced.

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 DICAS and LSR, respectively. Section 5 presents attacks against routing and their mitigation in LSR using DICAS. Section 6 analyzes the coverage and overhead of DICAS, 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 for securing the control traffic in wireless networks. These mechanisms can be broadly categorized into four classes – customized cryptographic primitives, protocols for path diversity, protocols that overhear neighbor communication, and protocols that use specialized hardware. The cryptographic primitives are also used as building block for protocols of the other three classes. In the context of ad hoc networks, HMAC and digital signatures [38] have been used to provide end-to-end authentication of the routing traffic [2],[5]. Intermediate node authentication of the source traffic has been achieved via authentic broadcasting techniques using digital signatures [23], hash trees [3], or μ-TESLA [4]. These protocols are restrictive and only capable of providing the traditional cryptographic guarantees, namely confidentiality and authenticity of the routing traffic. In addition, it is usually infeasible to apply them to sensor networks. The public key cryptography is beyond the capabilities of sensor nodes and the symmetric key based protocols proposed are too expensive in terms of node state and communication overhead.

The path diversity techniques increase route robustness by first discovering multipath routes [23],[30], [31] and then using these paths to provide redundancy in the data transmission between a source and a destination [29]. The data is encoded and divided into multiple shares sent to the destination via different routes. Many of these schemes are vulnerable to attacks that permit a node to assume multiple identities (such as the Sybil attack).

Mechanisms to overhear neighbor communication in a wireless channel have been used to minimize the effect of misbehaving nodes [19],[31],[35]-[37]. In the watchdog scheme [19], 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 blacklisting, 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 [35],[36], to build cooperative intrusion detection systems [37], or to discover multiple node-disjoint routes [31]. However, all these protocols use the communication overhearing as an existing service without studying its feasibility, requirements, limitations, or performance in the resource-constrained sensor environment. Examples of the fourth class are [20],[21], the former called packet leashes uses either tight time synchronization or location awareness through GPS hardware and the latter uses directional antennas. These schemes are used to detect one form of control attack – the wormhole attack.

On the other hand, many secure sensor network routing protocols have also been introduced in the literature [6]-[11]. 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 [18]. Table 1 enumerates the protocols and their vulnerabilities.

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

Routing protocol name	Attacks
Directional diffusion ([6], [9])	iii, iv, v, vii
GPSR [8]	iii, vii
Minimum cost forwarding [10]	i, iv, v, vii
LEACH [11], PEGASIS [24]	v, vii
Rumor routing [12]	i, iii, iv, vii
SPAN [15]	iii, v

Few of the protocols mentioned above 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 due to environmental conditions.

3. Description of DICAS

DICAS consists of a set of two algorithms as primitives (Section 3.2) and two main modules - the *local monitoring module* (Section 3.3), and the *local response module* (Section 3.4).

3.1. System Model and Assumptions

Attack model: A malicious node can be either an external node that does not know the cryptographic keys, or an insider node, that possesses the keys. 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 using arbitrary collusion with other nodes. 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. Also, we assume that a finite amount of time is required from a node's deployment for it to be compromised. This time is called the *compromise threshold time* T_{CT}. We define the maximum time required for the first and second hop neighbor discovery protocol (Section 3.2) to complete as T_{ND}. Our assumption is that for a given node n_i, all its first and second hop neighbors are deployed within T_{CT}-$2T_{ND}$ of the deployment of n_i. This assumption implies that for a given node, no malicious node exists in its one or two hop neighborhood till its neighbor discovery protocol completes. Our protocol uses nodes for overhearing and monitoring traffic on links called guards (Section 3.3). We assume that the network has sufficient redundancy, such that the attacker can not compromise more than an application defined threshold of guards in a certain transmission range within a certain time. This means that any node in the network has some good 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. Also the functional rules of a node, such as cluster head and regular sensing node, may change. Finally, we assume a key management protocol, such as [26], is used to pre-distribute pair-wise keys in the network so 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, *X*, is any node that lies within the transmission range of *X*. 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 an authenticated reply to *A*, using the pair-wise shared key. For each reply received within a pre-defined timeout (T_{ROUT},), *A* verifies the authenticity of the reply and adds the responder to its neighbor list, R_A. Let $R_A = n_1, ..., n_p$ and $M = R_A||K_{commit(A)}$, where $K_{commit(A)}$ is the commitment key *A* uses to authenticate itself to its neighbors. Node *A* computes $P = M||K_{An_1}(M)||\ldots||K_{An_p}(M)$. Then *A* sends a one-hop broadcast of packet *P*. A node n_j that receives *P*, verifies *M* using K_{An_j}. If the message is correctly verified, n_j stores R_A (n_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 is secure in static wireless networks because of the system model assumptions on time to compromise a node and the deployment of a node and its neighbors.

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, 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 $K_{commit(S)}$. 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 himself. Therefore, the attacker can impersonate that source and generate valid packets. In such case, we revert to the well-known µTESLA authentication scheme [25] that countermeasure such attacks.

3.3. Local monitoring: Detection and Diagnosis

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

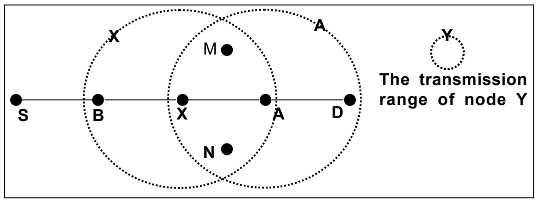

Figure 1: X, M, and N are guards of A over the link from X to A

For a node, say M, to be able to monitor a node, say A, two conditions are required: (i) each packet forwarder must explicitly announce the immediate source of the packet it is forwarding, and (ii) M must be a neighbor of both A and the previous hop from A, say X. The first condition is guaranteed universally by the routing protocol and therefore the second condition is the deciding criterion. In such a case, we call M a *guard* node of A over the link from X to A. In Figure 1, nodes M, N, and X are the guards of A over the link from X to A. For a link (i, j), the sender i is a guard node for node j. Information for each packet sent from X 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. In Figure 1, if a guard, say M, does not hear A forwarding a packet sent by X within τ, it accuses A of dropping or delaying the packet. If M hears A transmitting the packet within τ but detects a change in the packet's content or header, it accuses A of modifying the packet. If M hears A transmitting a packet, claiming that it was sent by X, but M does not have the corresponding incoming packet in its watch buffer, M accuses A of fabricating the packet. 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.

3.4. Local Response and Isolation

Detection and diagnosis is only the first step 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 appropriate response to isolate it from the network. The following local response algorithm is triggered by a guard node, say α, when a suspect malicious node, say A, is diagnosed.

1. When the *MalC(α,A)* crosses a threshold, $\boldsymbol{C_t}$, α revokes A from its neighbor list, and sends to each neighbor of A, say D, an authenticated alert message indicating A is a suspected malicious node. This communication is authenticated using the shared key between α and D to prevent false accusations. Alternately, if the clocks of all the nodes in the network are loosely synchronized, α can do authenticated local two-hop multicast as in [16] to inform the neighbors of A
2. When D receives the alert, it verifies its authenticity, that α is a guard to A, and that A is D's neighbor. It then stores ID_α in an *alert buffer* associated with A.
3. When D receives enough alerts, γ, about A, it isolates A by marking A's status as revoked in the

neighbor list. We call γ the *detection confidence index*.

4. After isolation, *D* does not accept any packet from or forward any packet to a revoked node.

In addition to removing the malicious nodes from the network, this module makes the response process fast since the detection knowledge does not need to propagate to all the nodes in the network. Also this module is lightweight in the number of messages (one to each neighbor of *A* only on malicious node 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 [28] protocol. However, LSR has significant differences 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 DICAS, LSR can deterministically detect and isolate nodes involved in launching these attacks. Section 6.1 provides detailed analysis of the detection and isolation coverage of control attacks in LSR with DICAS.

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 identity (ID_S), the destination 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 [22]. 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*.

1. [At *B*] Save "$REQ\|MAC_{K_{SD}}(REQ)$", and set T_r.
2. [At *B*] Save the identity of every neighbor that sends a *REQ* copy within T_r.
3. [At B] Select random copy of the *REQ*.
4. [At *B*] Store ID_S, ID_D, *SN*, and ID_W.
5. $B \xrightarrow{Broadcast} REQ\|MAC_{K_{SD}}(REQ)\|ID_W\| ID_B$

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 the pair-wise shared key (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* be the immediate previous hop from *A*.

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

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*.

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

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 the destination.

The route maintenance in LSR is triggered when a broken link is detected and a new route is discovered by using the above protocol for route discovery. In this respect, it is similar to AODV.

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 [23].

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. LSR supports secure discovery of these routes as a by-product of the local monitoring module of the underling DICAS protocol without incurring any additional overhead. 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 [28]. 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*.

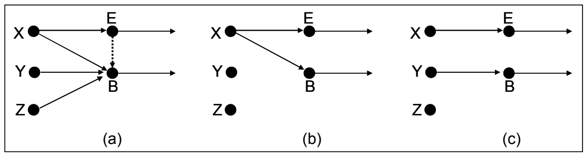

Figure 2: Example of node-disjoint routes.

An example is shown in Figure 2. Let *B* receive *REQ*s from nodes *X*, *Y*, and *Z*, and let *E* be a neighbor of *B* which also receives from *X*, and let the *REQ* from *X* 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 *X* 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 *X*, then *B* will drop *X*'s packet from its buffer. Thus *B* will not forward the *REQ* forwarded by *X*, The resulting paths are disjoint, Figure 2(c).

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* forward the *REQ* that has been forwarded by *A*. Let the two non-neighbor nodes, *X* and *Y*, receive and forward the *REQ* they get from *B*. The *REP* packet takes the reverse path, i.e. *B* gets the *REP* packets from *X* and *Y*. Without loss of generality, let the *REP* packets come from *X* then from *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 *X* 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 a set of 5 attacks that can be launched against a routing protocol and show how they can be detected in LSR with DICAS.

5.1. Route Traffic Manipulation

An attacker may attack the routing infrastructure by injecting false control packets, modifying the forwarded control packets, or replaying old authenticated control packets. This may result in creating routing loops, attracting network traffic, extending or shortening routes, generating false error messages, partitioning the network, or increasing the end-to-end delay.

Conjecture#1: DICAS detects any injection, alteration, or replaying of the routing traffic in LSR.

Proof sketch: The end-to-end authentication prevents a malicious node from injection or alteration of the *REQ* and the *REP* packets. The increasing sequence number associated with each *REQ* and *REP* prevents the replay attack.

5.2. ID Spoofing and Sybil Attacks

In this attack, an attacker presents one (ID spoofing) or more (Sybil attack) spoofed identities to the network [13]. Those identities could either be new fabricated identities or stolen identities from legitimate nodes. The Sybil attack can have many adverse impacts, such as on multipath routing [14] and collaborative protocols that use aggregation and voting [40].

Conjecture#2: In LSR with DICAS, malicious node ID spoofing or Sybil attack attempts can be easily detected.

Proof sketch: (i) The single 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 authenticated source 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 prevents a forwarding node from spoofing a neighbor's identity. As shown in Figure 1, if *A* receives a packet from *X*, then *A* can not forward the packet claiming that it is being forwarded by one of its neighbors, say *M*. None of the guards of *M* over the link from *X* to *M* overhear such a packet; also the guards of *A* over the link from *X* to *A* accuse *A* of not forwarding the packet.

5.3. Wormhole Attack

In the wormhole attack [20],[21] 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 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. Later, he can use this to perform traffic analysis or selectively drop data traffic.

The wormhole attack can affect network routing, data aggregation and clustering protocols, and location-based wireless security systems. Finally, it is worth noting that the wormhole attack can be launched even without having access to any cryptographic keys or compromising any legitimate node in the network.

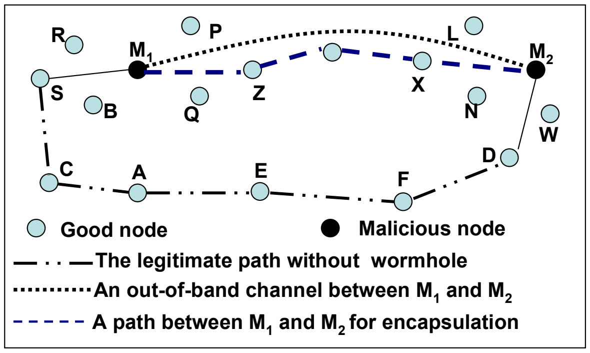
R
P
L
M_1
M_2
S
B
Q
Z
X
N
W
D
C
A
E
F
Good node
Malicious node
The legitimate path without wormhole
An out-of-band channel between M_1 and M_2
A path between M_1 and M_2 for encapsulation

Figure 3: A wormhole attack scenario

Conjecture#3: DICAS detects and isolates malicious nodes that are involved in a wormhole attack.

Proof sketch: 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 *X*. 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 *X* to M_2 (*X*, *N*, and *L*) detect M_2 as fabricating the route request since they do not have the information for the corresponding packet from *X* 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.

5.4. Sinkhole

In the sinkhole attack [18], a malicious node manages to attract routes from many nodes to go through it thus acting as a "sinkhole". This attack typically works by making the malicious node look especially attractive for the surrounding nodes, for example, by claiming a short or a fast route to the destination. If the attacker succeeds, he can launch data traffic attacks and can prevent the discovery of other legitimate routes.

Conjecture#4: DICAS detects any malicious attempts to establish a Sinkhole in LSR.

Proof sketch: In DICAS end-to-end authentication and local monitoring prevent the sinkhole attack. An intermediate node does not accept any routing traffic from a non-neighbor nor does it forward any routing traffic to a non-neighbor. Also a destination node does not accept any routing traffic from a source node unless that traffic is authenticated using the shared key.

5.5. Rushing Attack

In the rushing attack [22], an adversary who receives a *REQ* rushes to broadcast it in an attempt to make the *REQ* forwarded by him to be the first to reach all the neighbors of the destination. If the attacker succeeds, then any route discovered by this rushed *REQ* includes a hop through the attacker.

Conjecture#5: LSR mitigates the rushing attack.

Proof: The design of the route discovery module of LSR implements a variant of the rushing attack prevention protocol (*RAP*) as proposed in [22]. 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. The waiting stops the rushing of the attacker and the random selection reduces the likelihood of selecting a route through the attacker node. Also the multiple node-disjoint route creation protocol prevents a single malicious node from affecting multiple routes between a source-destination pair.

6. DICAS analysis

6.1. Coverage analysis

In this section, we quantify the probability of missed detection and false detection of a generic control 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. 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*. The value of *x* follows a random variable with probability density function of $f(x) = 2x/r^2$ with range (0,*r*). This follows from the assumption of uniform distribution of the nodes.

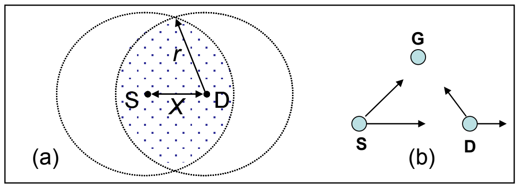

Figure 4: (a) The area where a node can guard the link between S and D; (b) Illustration for detection accuracy

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 $Area(x) = 2r^2\cos^{-1}\left(\frac{x}{2r}\right) - (2x)\sqrt{r^2 - \frac{x^2}{4}}$. The minimum value of *Area(x), Area*$_{min}$, is when *x* = *r*. Therefore, the minimum number of guards is $g_{\min} = Area_{\min} d = 0.36r^2 d$. The expected value of *Area(x)*

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

$$= \left(\frac{2\pi}{3} - \frac{1}{2}\right)r^2 \approx 1.6r^2$$

Therefore, the expected number of guards is $g = E[Area(x)]d = \lfloor 1.6r^2 d \rfloor$. The number of neighbors of a node is given by $N_B = \pi r^2 d$.

$$g = \left(\frac{2}{3} - \frac{1}{2\pi}\right)N_B \approx \lfloor 0.51 N_B \rfloor \quad \text{——} \quad \text{(I).}$$

Now, as in [33] where IEEE 802.11 was analyzed, we assume that each packet collides on the channel with a constant and independent probability, P_C. As shown in Figure 4(b), a guard *G* will not detect a packet sent by *D*, claiming it was received from *S*, if *G* experienced a collision at the time that *D* transmits. Thus, the probability of missed detection is P_C. Assume that μ packet attacks (fabrication, modify, drop, etc.) occur within a certain time window, *T*. 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. Then, the alert probability at a guard is given by $P_{\beta|\mu} = \sum_{i=\beta}^{\mu}\binom{\mu}{i}(1-P_C)^i (P_C)^{\mu-i}$. Thus, assuming independence of collision events among the different guards, the probability that at least γ of the guards generate an alert is given by

$$p_{\geq\gamma} = \sum_{i=\gamma}^{g}\binom{g}{i}\left(P_{\beta|\mu}\right)^i\left(1-P_{\beta|\mu}\right)^{g-i} = \frac{B(P_{\beta|\mu}, \gamma, g-\gamma+1)}{B(\gamma, g-\gamma+1)}$$

$$= \frac{g!}{(\gamma-1)!(g-\gamma)!}\int_0^{P_{\beta|\mu}} u^{\gamma-1}(1-u)^{g-\gamma}du$$

where, $B(\gamma, g-\gamma+1)$ is the Beta function and $B(P_{\beta|\mu}; \gamma, g-\gamma+1)$ is the incomplete Beta function.

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 = 0.05 at N_B = 3. Thereafter, P_C is assumed to increase linearly with the number of neighbors. 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.

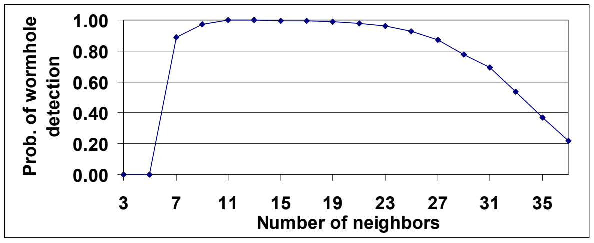

Figure 5: Probability of attack detection

Figure 9 shows, for the same μ, β, and P_C as in Figure 5, the probability of attack detection as a function of γ when N_B = 15 and *M* = 2. As γ increases, the probability decreases. As shown in Figure 4(b), a false alarm occurs when D receives a packet sent from S, while G does not receive that packet, and later, G receives the corresponding packet forwarded by D. Thus, the probability of false alarm is $P_{FA} = P_C(1-P_C)^2$. 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 β or more packets are falsely suspected as wrong packets. This 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

$$p_{FA\geq\gamma} = \sum_{i=\gamma}^{g} \binom{g}{i} \left(P_{FA(\beta|\mu)}\right)^i \left(1-P_{FA(\beta|\mu)}\right)^{g-i}$$

$$= \frac{\beta(P_{FA(\beta|\mu)}, \gamma, g-\gamma+1)}{\beta(\gamma, g-\gamma+1)} = \frac{g!}{(\gamma-1)!(g-\gamma)!} \int_0^{P_{FA(\beta|\mu)}} u^{\gamma-1}(1-u)^{g-\gamma} du$$

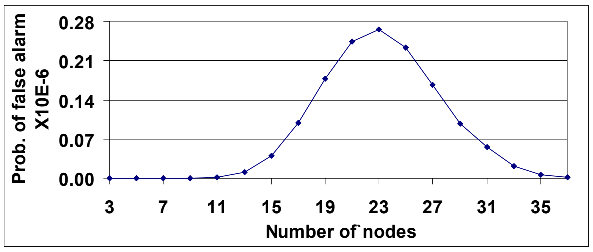

Figure 6: Probability of false alarm

Figure 6 shows the probability of false alarm as a function of the number of nodes for the same parameters as in Figure 5. The non monotonic nature of the plot can be explained as follows. As the number of neighbors increases, so does the number of guards. Initially, this increases the probability that at least γ guards miss the packet from S to the guard but not from D to the guard, leading to false detection at these γ guards. Beyond a point, however, the increase in the number of neighbors increases the collision probability. This increases the probability that both of these packets are missed at the guard and thus does not lead to false detection. The worst case false alarm probability is still negligible (less than 0.3×10^{-6}).

6.2. Cost Analysis

In this section, we show the memory, the computation, and the bandwidth overhead of DICAS to evaluate its suitability to resource-constrained environments.

Memory overhead: Each node needs to store a neighbor list, a commitment key of each first hop neighbor, its own commitment string, a watch buffer, and an alert buffer. Assuming that the identity of a node is 2 bytes and reusing the notation from the previous subsection, the size of the neighbor list is $NB_L = \pi r^2 d$ entries. Each entry in NB_L uses 3 bytes; 2 for identity of the neighbor and 1 for the *MalC* associated with that neighbor. Each first hop entry in NB_L requires 20 more bytes (e.g. SHA-1[38],[25],[39]) for the storage of the commitment key. So the total NB_L storage, $NB_{LS} = 25(\pi r^2 d)^2$. Also, a commitment string of length l requires $20l$ bytes. For example, NB_{LS} and the commitment string use less than a kilobyte when $N_B = 10$ and $l = 20$. The alert buffer has γ number of 2 byte entries. If we monitor the *REP* packets of LSR, then the watch buffer size depends on the average number of hops between a source-destination pair (h), the frequency of route establishment aggregated over the network (f) and the node density (d). We calculate the average number of nodes involved in monitoring a *REP*, $N_{REP} = 2r^2(h+1)d$, by creating a rectangular bounding box of dimensions $((h+1)\ r \times 2r)$ containing the nodes that may overhear the *REP* sent from A to B. This is an overestimate since we use a square that circumscribes the circular transmission range. Thus, given N as the total number of nodes in the network, each node is involved in monitoring at most $(N_{REP}/N)f$ route replies per unit time. For example, if $N = 100$ nodes, $h = 4$ hops, and $f = 1$ route every 4 time units, then $N_{REP} = 17$, and each node monitors 4 route replies every 100 time units. Because the time τ for which the packet is kept in the watch buffer is relatively small being determined by the MAC layer delay for acquiring the channel, a watch buffer size of 4 entries is sufficient (for $\tau\leq10$). If we also monitor the *REQ*, then each node is involved in monitoring $f+(N_{REP}/N)f$ packets. This requires each node to monitor 4 packets every 16 time units. Again a 4-entry watch buffer is sufficient. 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*).

Computation and bandwidth overhead: Each monitored *REP* (*REQ*) requires one lookup for the current source and destination in the neighbor list, adding an entry to the watch buffer (incoming) or deleting an entry from the watch buffer (outgoing). Since the size of the watch buffer and the neighbor list structure are relatively small, the computation time required for these operations is negligible. For example, a lookup in a 100 entry buffer takes the MICA mote with an Atmega128 4 MHZ processor, about 2μ seconds. The bandwidth overhead is incurred after deployment of a node for neighbor discovery and in the case of wormhole detection for informing the neighbors of the detected node. This is therefore a negligible fraction of the total bandwidth over the lifetime of the network.

7. Simulation Results

We use the *ns-2* simulator [34] to simulate a data exchange protocol over LSR, individually without DICAS (the *baseline*) and with DICAS. We distribute the nodes randomly over a square area with a fixed average node density. Thus, the length of the square varies (80m to 204 m) with the number of nodes (20-150). We first simulate the wormhole attack using out-of-band direct channels between the colluding 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 using an exponential random distribution with inter-arrival rate of 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. The experiment parameters are presented in Table 2. The results are obtained by averaging over 30 runs. For each run, the malicious nodes are chosen at random so that they are more than 2 hops away from each other.

Table 2: Input parameter values

Parameter	Value	Par	Value
Tx Range (r)	30 m	γ	2-8
N_B	8	μ	100 ms
$TOut_{Route}$	50 sec	M	0-4
τ, N_r	0.05 s, 5	β	5
Channel BW	40 kbps	ξ	5m s

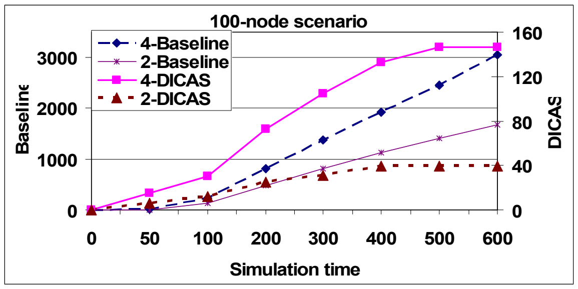

Figure 7: Cumulative number of dropped packets

Figure 7 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 sec after the start of the simulation. Since the numbers are vastly different in the baseline and with DICAS, they are shown on separate Y-axes. 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 DICAS, 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 at 75 sec, due to the cached routes that contain the wormhole and continue to be used till route timeout occurs.

Figure 8 shows a snapshot, at simulation time of 2000 sec, 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 DICAS. 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 DICAS, they would tend 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.

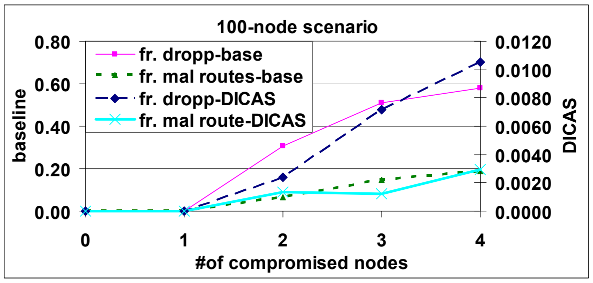

Figure 8: Fraction of dropped packets & malicious routes

Figure 9 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, though it is very small (less than 30 s) even at the right side of the plot.

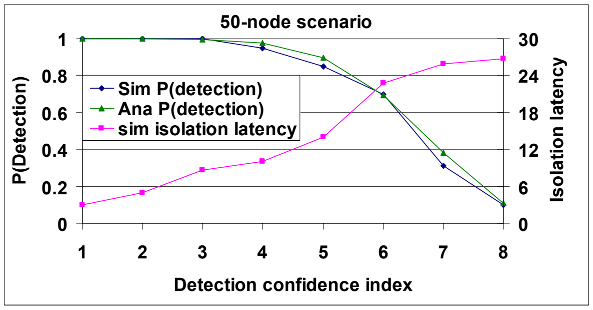

Figure 9: Detection probability and isolation latency

Next, we simulate the 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 [32], and LSR with DICAS. In the ideal search, the topology of the entire network is known to the source which uses 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 10 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 performs almost identically. In a malicious scenario, 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 10 shows that LSR with DICAS 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.

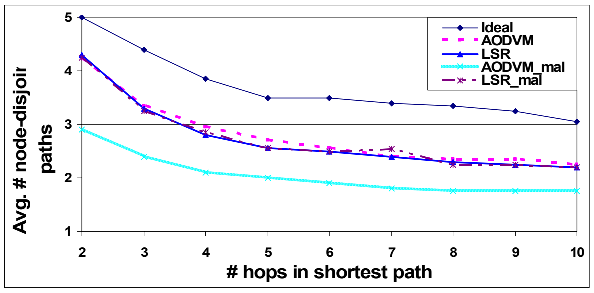

Figure 10: Average number of node-disjoint paths in ideal case, AODVM, and LSR

8. Conclusion

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

We note that although designed for static networks, DICAS 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. Two existing protocols can be used to enable secure neighbor discovery in mobile wireless networks: (i) *directional-antenna-based neighbor detection* [21], which uses the knowledge of the direction of a received packet and the direction of the corresponding transmission and (ii) *propagation delay based neighbor detection* [22], which uses packet delay of certain control packets to measure the distance to a neighbor. These protocols, however, are not well-suited to sensor networks because of the non-negligible communication overhead and the expensive hardware. As future work we are investigating secure neighbor discovery protocols appropriate for mobile networks.

9. References

[1] L. Zhou and Z. Haas, "Securing ad hoc networks," IEEE Network Magazine, vol. 13, no. 6, November/December 1999.

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

[3] 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.

[4] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," MobiCom 02, pp. 12-23.

[5] 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.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," MobiCom 2000.

[7] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," Mobicom, 2001.

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

[9] 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.

[10] 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.

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

[12] D. Braginsky and D. Estrin, "Rumor routing algorithm for sensor networks," WSNA 2002.

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

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

[15] Y. Xu, J. Heidemann, and D. Estrin, "Geography-informed energy conservation for ad hoc routing," MobiCom, 2001.

[16] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," NDSS 2001.

[17] D. Johnson, D. Maltz, and J. Broch, "The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks," Ad Hoc Networking, C. Perkins, Ed. Addison-Wesley, 2001.

[18] C. Karlof and D. Wagner, "Secure Routing in Sensor Networks: Attacks and Countermeasures," SNPA 2003.

[19] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," MobiCom 2000.

[20] Y. C. Hu, A. Perrig, and D.B. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," IEEE InfoCom 2003.

[21] L. Hu and D. Evans, "Using Directional Antennas to Prevent Wormhole attacks," NDSS 2004.

[22] Y. C. Hu, A. Perrig, and D. Johnson, "Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols," WiSe 2003.

[23] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. Belding-Royer, "A Secure Routing Protocol for Ad hoc Networks," ICNP 02.

[24] S. Lindsey and C. Raghavendra, "PEGASIS: power-efficient gathering in sensor information systems," IEEE Aerospace Conference, 2002.

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

[26] D. Liu and P Ning, "Establishing Pair-wise Keys in Distributed Sensor Networks," CCS 2003.

[27] F. Zhao and L. Guibas (Eds.), "A Performance Evaluation of Intrusion-Tolerant Routing in Wireless Sensor Networks," IPSN 2003, pp. 349-364, 2003.

[28] 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), pp. 90-100, February 1990.

[29] P. Papadimitratos and Z.J. Haas, "Secure Message Transmission in Mobile Ad Hoc Networks," WiSe 2003.

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

[31] 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):339-349.

[32] Z. Ye, S. V. Krishnamurthy, S. K. Tripathi, "A Framework for Reliable Routing in Mobile Ad Hoc Networks," IEEE InfoCom 2003.

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

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

[35] 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.

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

[37] Y. Huang and W. Lee, "A Cooperative Intrusion Detection System for Ad Hoc Networks," SASN 2003.

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

[39] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical En-route Detection and Filtering of Injected False Data in Sensor Networks," InfoCom 2004.

[40] M. Krasniewski, P. Varadharajan, B. Rabeler, S. Bagchi, Y. C. Hu, "Tibfit: Trust Index Based Fault Tolerance for Arbitrary Data Faults in Sensor Networks," To appear in the International Conference on Dependable Systems and Networks (DSN) 2005.