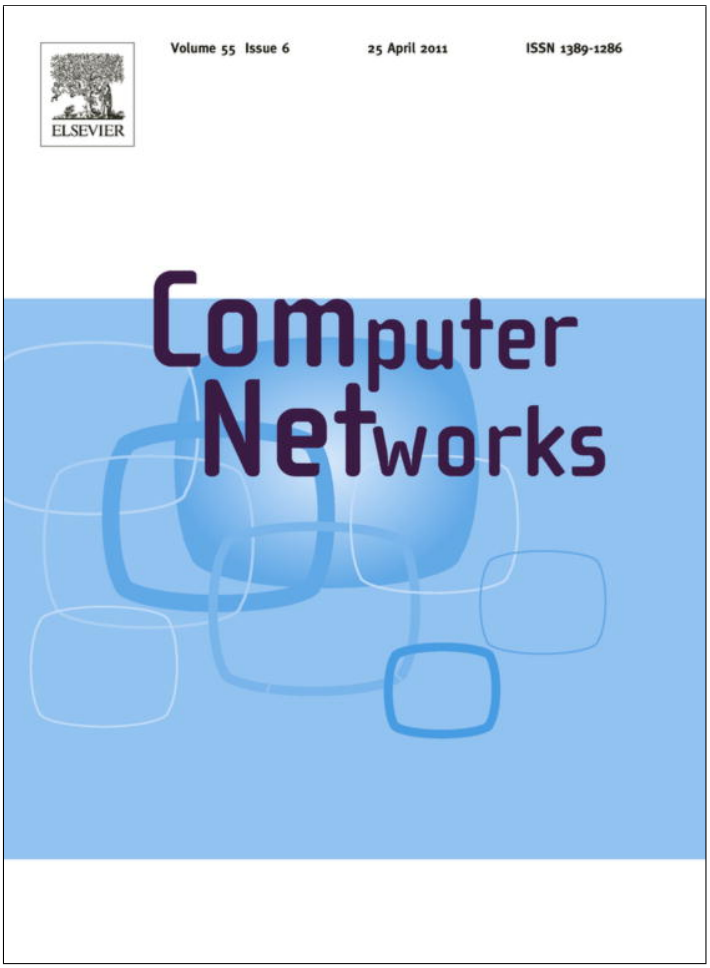


Computer Networks 55 (2011) 1229–1241

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Secure neighbor discovery through overhearing in static multihop wireless networks ☆

Srikanth Hariharan [a,*], Ness B. Shroff [b], Saurabh Bagchi [c]

[a] Department of ECE, The Ohio State University, 205 Dreese Laboratories, 2015 Neil Avenue, Columbus, OH 43210, United States
[b] Departments of ECE and CSE, The Ohio State University, 205 Dreese Laboratories, 2015 Neil Avenue, Columbus, OH 43210, United States
[c] School of ECE, Purdue University, 465 Northwestern Avenue, West Lafayette, IN 47907, United States

ARTICLE INFO

Article history:
Received 20 May 2010
Received in revised form 16 October 2010
Accepted 27 October 2010
Available online 7 December 2010
Responsible Editor: I.F. Akyildiz

Keywords:
Neighbor discovery
Security
Wireless networks
Overhearing

ABSTRACT

In wireless ad-hoc and sensor networks, neighbor discovery is one of the first steps performed by a node upon deployment and disrupting it adversely affects a number of routing, MAC, topology discovery and intrusion detection protocols. It is especially harmful when an adversary can convince nodes that it is a legitimate neighbor, which it can do easily and without the use of cryptographic primitives. In this paper, we develop a secure neighbor discovery protocol, SEDINE, for static multihop wireless networks. We prove that, in the absence of packet losses, without using any centralized trusted node or specialized hardware, SEDINE prevents any node, legitimate or malicious, from being incorrectly added to the neighbor list of another legitimate node that is not within its transmission range. We provide simulation results to demonstrate the efficacy of SEDINE, in the presence of packet losses.

1. Introduction

Wireless ad-hoc and sensor networks are increasingly being used in a number of commercial, industrial and military applications for data monitoring. The ability of nodes to self-configure (without the assistance of a powerful base station) allows these nodes to be deployed in inhospitable and hostile environments that need to be monitored. Security of the monitored data could be of great concern. Neighbor discovery is one of the first steps performed by a node before it starts monitoring. Neighbor discovery, as the name suggests, is the process of identifying neighbor nodes. A neighbor of a node X is defined as one that is within the radio communication range of X.

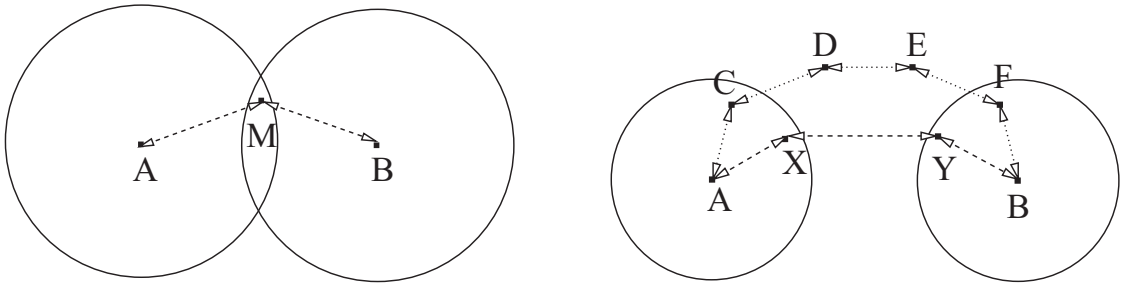
An adversary intending to disrupt the neighbor discovery protocol will try to make two non-neighboring nodes believe that they are neighbors, or will prevent two neighboring nodes from becoming neighbors. By launching the former attack, the adversary can in turn attack protocols that need accurate neighbor information. For example, an adversary can attack routing protocols such as AODV [20] and DSR [9] by launching a wormhole attack. In a wormhole attack, malicious nodes can either falsely convince two non-neighboring nodes that they are within communication range, or falsely convince the nodes that the malicious nodes belong to the best possible route between the source and the destination. This attack can be launched even without requiring the cryptographic keys in the network. The adversary, after inserting itself in the false routes, can control the packets sent over those routes, e.g., by selectively dropping packets and crypt-analyzing them.

We now illustrate the importance of secure neighbor discovery. Consider two legitimate non-neighboring nodes A and B, and a malicious node M that is within the communication range of both A and B (Fig. 1(a)). If the neighbor discovery protocol simply consists of broadcasting a HELLO packet and receiving a response for the HELLO packet (as is

☆ This work has been supported in part by the NSF grants 0721236, 0626830, 0831060, and CNS-0626830.

* Corresponding author. Tel.: +1 859 285 0126.
E-mail addresses: harihars@ece.osu.edu (S. Hariharan), shroff@ece.osu.edu (N.B. Shroff), sbagchi@purdue.edu (S. Bagchi).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.10.021



(a) M fools A and B into believing that they are neighbors.

(b) X and Y fool A and B, which are far away, into believing that they are neighbors.

Fig. 1. Insecure neighbor discovery.

typical [20,9,10]), the malicious node *M* can relay the packet sent by *A* to *B* and vice versa. *A* and *B* will then believe that they are neighbors. If there exists two malicious nodes *X* and *Y* with powerful antennas or out-of-band channels, they can even make nodes that are multiple hops away from each other to believe that they are neighbors (Fig. 1(b)). By establishing these false links, the malicious nodes can insert themselves on the path between *A* and *B*. So, while the path that *A* should take to send a packet to *B* be through the legitimate nodes *C*, *D*, *E* and *F*, *A* will use the route *A–X–Y–B*.

Not only is it important to prevent two legitimate non-neighboring nodes from becoming neighbors, but it is also important to prevent a legitimate node from adding a malicious non-neighboring node to its neighbor list. For example, in Fig. 1(b), let *X* and *Y* be compromised malicious nodes and let *A* and *B* be legitimate nodes. *B* can be fooled into believing that *X* is its neighbor (since *Y* can relay packets between *B* and *X*). Hence, when *B* wants to send a packet to *A*, *B* will believe that the route is *B–X–A* while the actual route is *B–Y–X–A*. This may make the route through *X* and *Y* look attractive to *B*. Thus, false routes can be established. *The goal of this work is to prevent a legitimate node from adding a node that is not within its communication range, to its neighbor list.*

The disadvantages of insecure neighbor discovery are now apparent. What is also significant is that if neighbor discovery is made secure, the afore-mentioned wormhole attack can be effectively mitigated by building on the guarantee that neighbor information is accurate [10,7]. This is the key motivation for working on this problem.

We now overview related work. Typically, most work (e.g., [10]) have assumed that neighbor discovery is secure by reasoning that since neighbor discovery takes a very short time (typically a few seconds) it is unlikely for a node to get compromised before neighbor discovery is completed. While this may be true, the adversary *need not compromise a node* to disrupt the neighbor discovery protocol. An external malicious node, i.e., a node that does not possess cryptographic keys, can relay packets between legitimate non-neighboring nodes and make them believe that they are neighbors. The assumption that no compromised node exists during neighbor discovery also does not hold when nodes are incrementally deployed.

A number of protocols have also managed to move away from this assumption. Some of these protocols rely on timing information [8,5,14]. Eriksson et al. [5] also uses public key cryptography which may not be suitable for sensor networks due to high computation and memory requirements. These protocols use bounds on the delay between sending a message to the responder and receiving a message from the responder to determine whether the responder is actually within communication range. The main issue with these protocols is that characterizing delay is a hard problem in wireless networks due to interference, congestion, and link errors. Therefore it is hard to prove that timing information actually guarantees secure neighbor discovery. In fact, Poturalski et al. [22] show that timing information alone cannot guarantee secure neighbor discovery.

Another class of protocols rely on specialized hardware such as directional antennas [7] or advanced physical layer features [23]. The directional antenna protocol [7] substantially degrades network connectivity and does not consider framing attacks. Rasmussen and Capkun [23] proposes a technique called sensor fingerprinting in which a sensor can be identified based on the signal it transmits. The signal thus acts as a fingerprint for the sensor. It is unclear whether this approach would be practically feasible.

A location-based technique proposed by Zhang et al. [26] assumes that nodes can accurately estimate their location. But the location estimation protocol is itself subject to attacks [15,11,14]. Also, it is not a good idea to incorporate communication range into the protocol design since the communication range usually has a skewed pattern in practice. Lee and Choi [12] have proposed an approach for securely discovering second hop neighbors after assuming that the first hop neighbors can be securely discovered.

What is commonly lacking in many of these protocols is that they do not provide any *provable* security guarantees for neighbor discovery. Recently, Papadimitratos et al. [19] have also explained the importance of providing security guarantees for neighbor discovery in their survey paper. Another interesting paper by Maheshwari et al. [17] provides a theoretical foundation using connectivity information to determine false links in wireless networks. However, their scheme works only when the wormhole is sufficiently long.

We develop a new protocol called SEDINE to achieve secure neighbor discovery. It does not require specialized hardware and relies on the overhearing capability of nodes to detect whether a packet is being relayed. Solutions that involve overhearing have been proposed to increase the reliability of data reports sent by sensor nodes [16,24] and also to mitigate the effects of security attacks such as the wormhole attack [10]. The main contributions of this paper are as follows:

- We develop a *provably secure* neighbor discovery protocol that *does not require any specialized hardware such as directional antennas or highly accurate time measurement*.
- *We analytically guarantee that no two non-neighboring legitimate nodes can be fooled into becoming neighbors, in the absence of packet losses*. Further, for the problem of preventing two non-neighboring legitimate nodes from believing that they are neighbors, we show that guaranteeing security against packet relaying attacks is equivalent to guaranteeing security against all attacks.
- We quantify our results by taking packet losses into account and show through simulations that even in this scenario, the fraction of non-neighboring nodes that believe that they are neighbors is significantly smaller when using SEDINE than when using the insecure protocol.
- We extend our protocol to allow for incremental node deployment and also explain how we can improve some of the security properties of SEDINE by combining it with existing protocols (such as [17]).

The rest of this paper is organized as follows: we present the system model and detail our assumptions in Section 2. Section 3 describes the neighbor discovery protocol for static multi-hop wireless networks and an extension to allow for incremental node deployment. In Section 4, we provide analytical results with respect to security and connectivity. In Section 5, we discuss our analytical results and provide examples to show the effectiveness of SEDINE. In Section 6, we present our simulation results. Finally, we conclude our paper and discuss open problems in Section 7.

2. System model and assumptions

2.1. System model

We assume that all links are bi-directional, i.e., if node *A* hears node *B*, then node *B* also hears node *A*. We assume omni-directional antennas on nodes. SEDINE does not require nodes to have specialized hardware such as GPS devices or directional antennas. Further, SEDINE does not require a trusted base station or time synchronization between nodes. SEDINE requires a pair-wise key management protocol (for example, key pre-distribution techniques as presented in [3,4,13] that will allow any two nodes to establish a secure communication channel between them). Key management protocols are fundamental in securing wireless networks. Not only do neighbor discovery protocols like SEDINE require key management but also any protocol that requires secure communication between nodes. Therefore, we do not delve deeper into key management in this paper since our focus is on securing neighbor discovery. We initially assume that all nodes are static and that there is no incremental deployment. We later explain in Section 3 how SEDINE can be extended to allow incremental deployment. In our model, we allow packet losses to occur due to link errors or collisions.

2.2. Attack model

Malicious nodes may either be external nodes (that do not possess the cryptographic keys) or insider nodes (that have been compromised by the adversary). We relax the general assumption that no malicious nodes exist during the neighbor discovery process and instead assume that malicious nodes (both external and compromised) do not possess specialized hardware such as out-of-band channels or power controlled transmission (including using directional antennas) during neighbor discovery. Note that this assumption is not typically required in many of the wormhole detection papers. We reiterate that this paper is concerned with neighbor discovery and not with wormhole detection. We recognize that our neighbor discovery protocol is shown to provide provable guarantees under a certain class of attacker models. While these attacker models are not required for various security related works (e.g., wormhole detection), these works circumvent the problem by assuming a neighbor discovery mechanism that cannot provide any *provable* security guarantees. Thus, this work can be seen as foundational to the development of other more sophisticated security related works that rely on secure neighbor discovery.

The adversary can try to make two non-neighboring nodes believe that they are neighbors. Malicious nodes, both compromised and external, can collude with other malicious nodes. Essentially, the main intention of a malicious node would be to expand its neighbor list as well as the neighbor lists of its neighbors. By doing so, the adversary can establish false routes by possibly launching a wormhole attack in the future. *We do not consider attacks that prevent two neighboring nodes from becoming neighbors. Examples of such attacks include denial of service attacks, physical layer jamming attacks, and physical destruction of nodes.*

3. The neighbor discovery protocol

In this section, we develop a new protocol called SEDINE, for secure neighbor discovery in static multi-hop wireless networks. SEDINE consists of two phases:

1. The Neighbor Discovery Phase
2. The Neighbor Verification Phase

We first provide an overall idea of the protocol. During the Neighbor Discovery Phase, the *expected neighbors* of a node are discovered. The *expected neighbor list* of a node

A consists of nodes that are its actual neighbors and also nodes that are not within the communication range of A but have been made to believe that A is their neighbor by a malicious node in the network. During the Neighbor Verification Phase, we propose a technique to filter out those nodes that are not within the communication range of A from the *expected neighbor list* of A using *verifiers*. A *verifier* of a link $A \leftrightarrow B$ is a node that is in the *expected neighbor list* of both A and B. In order to perform link verification, each node requires the following.

- Each node needs to find its expected neighbors.
- Each node needs to know the expected neighbors of each of its expected neighbors.

Each node then determines the verifiers for each of its links and also determines the links for which it is a verifier. Each verifier of a link, during the Neighbor Verification Phase, checks whether a packet sent on that link is being relayed to the next hop. Depending on whether the packet is being relayed, each verifier takes an independent decision on whether the link is legitimate. Verifiers then exchange their responses between themselves and between the source and destination.

3.1. The Neighbor Discovery Phase

3.1.1. Determining the expected first hop neighbors

In this phase, each node determines the nodes that claim to be its first hop neighbors. Upon deployment, each node broadcasts a *HELLO* packet and its node ID. Every node that hears this *HELLO* packet sends back its ID and a reply containing a nonce which is authenticated using the key that is shared between the nodes. This key, for example, could be pre-distributed between the two nodes [3,4,13]. The initiating node accepts all replies that arrive within a timeout and then authenticates itself to each of its neighbors by sending a hash value of the nonce that it received from them and adds them to its neighbor list. We call this neighbor list as the *expected neighbor list*. This list may include nodes that are not actually within the communication range of the initiating node. This is because a malicious node or a set of malicious nodes could have relayed these packets between the initiating node and another node that is not within the communication range of the initiating node to make them believe that they are neighbors. The subsequent Neighbor Verification Phase provides a mechanism to filter out the non-neighboring nodes from the *expected neighbor list* of a node.

3.1.2. Determining the expected second hop neighbors

Once each node has determined its expected list of neighbors, it needs to know the expected neighbors of each of the nodes in this list to determine the *verifiers* of each of its own claimed links. The *verifiers* will be used in the Neighbor Verification Phase to decide whether two nodes are actually neighbors. We now propose a simple technique to determine the verifiers.

Each node generates a random key, K, and uses this key to encrypt its expected neighbor list and the list of hash values of the nonces that were used when discovering neighbors (Table 2, Steps 6 and 7). Each node then broadcasts this encrypted expected neighbor list. After broadcasting, each node waits until a timeout to receive the corresponding expected neighbor list of each of its expected neighbors. Nodes that do not send their expected neighbor list within the timeout period are discarded from the expected neighbor list of the initiating node. When the timeout expires, each node broadcasts key K and the set of discarded nodes. At this point, each node knows its expected neighbors and the expected neighbors of each of its expected neighbors.

Note that delaying the process of revealing the key K serves two purposes.

(1) It ensures that packets exchanged between legitimate nodes cannot be tampered by a malicious node. We provide examples for this in Section 5.
(2) It utilizes the broadcast nature of the wireless medium, thus saving energy spent on communication.

Here, it is interesting to note that if public key cryptography can be used, this step becomes much simpler. A node can simply sign and broadcast its expected neighbor list. Nodes that receive this expected neighbor list can verify the signature, and thus authenticated broadcast can be performed.

This protocol can be extended so that a node can also determine the verifiers of the link between its expected neighbor (say X) and any expected neighbor of X. By doing this, the node can verify whether the nodes that X claims to be its neighbors, are actually the neighbors of X. This is important for protocols that require accurate information of second-hop neighbors as well. We now explain this extension.

After having received the expected neighbor list of each of its expected neighbors, instead of revealing the key K and the dropped neighbors, each node generates a new key K'. The expected neighbor lists acquired are now encrypted with K' and broadcasted as described in Steps 10 and 11 in Table 2. If a node does not send this set of expected neighbor lists within a timeout, it will be dropped from the corresponding expected neighbor list. After receiving these two lists, each node reveals keys K, K', the dropped neighbors, and the keys revealed by each of its expected neighbors.

We now provide some notations in Table 1, and summarize the Neighbor Discovery Phase in Table 2.

At the end of this phase, each node S knows $\widetilde{N}(S)$, $\widetilde{N}(T)\ \forall T \in \widetilde{N}(S)$ and $\widetilde{N}(V)\ \forall V \in \widetilde{N}(T)$.

Table 1
Notations.

$\widetilde{N}(S)$	The expected neighbor list of node S
$K_{X,S}$	The key shared by node X and node S
$h(\cdot)$	A hash function
$K_{S,Bcast}$	A key used by S to broadcast
T_{out}	A timeout

Table 2
The Neighbor Discovery Phase.

Determining the one hop expected neighbors
1. $S \rightarrow$ One hop broadcast: HELLO, ID_S
2. $X \rightarrow S$: ID_X, $K_{X,S}$(HELLO reply, nonce $N_{X,S}$)
3. $S \rightarrow X$: $K_{X,S}$(Ack, $h(N_{X,S})$)
4. S: Adds the ID of X to its expected neighbor list, $\widetilde{N}(S)$
5. S: Repeats steps 2, 3 and 4 for every HELLO reply received

Determining the expected two hop neighbors
6. S: Generate key $K_{S,Bcast}$
7. $S \rightarrow$ One hop broadcast: $K_{S,Bcast}(ID_S, \{(h(N_{X,S}), X)\ \forall X \in \widetilde{N}(S)\})$
8. S: Wait for $min(T_{out}, \widetilde{N}(T)\ \forall T \in \widetilde{N}(S))$
9. S: Drop nodes that do not send their expected neighbor list within T_{out} from $\widetilde{N}(S)$
10. S: Generate key $K'_{S,Bcast}$
11. $S \rightarrow$ One hop broadcast: $K'_{S,Bcast}(ID_S, \{(h(N_{T,S}), K_{T,Bcast}(\widetilde{N}(T)))\ \forall T \in \widetilde{N}(S)\})$
12. S: Wait for $min(T'_{out}, \widetilde{N}(V)\ \forall T \in \widetilde{N}(S)$ and $\forall V \in \widetilde{N}(T))$
13. S: Drop nodes that do not send their neighbors' neighbor list within T'_{out} from $\widetilde{N}(S)$
14. $S \rightarrow$ One hop broadcast: $K_{S,Bcast}(ID_S$, Dropped Neighbors)
15. S: Wait until a timeout to receive $K_{T,Bcast}(ID_T$, Dropped Neighbors) $\forall T \in \widetilde{N}(S)$
16. $S \rightarrow$ One hop broadcast: $K_{S,Bcast}$
17. S: Wait to receive $K_{T,Bcast}\ \forall T \in \widetilde{N}(S)$
18. $S \rightarrow$ One hop broadcast: $K'_{S,Bcast}, K_{T,Bcast}\ \forall T \in \widetilde{N}(S)$

3.2. The Neighbor Verification Phase

Once each node has completed the Neighbor Discovery Phase, it can determine the verifiers for each of its links. Furthermore, each node can also determine the links for which it is a verifier. For example, consider two nodes A and B that are in the expected neighbor lists of B and A, respectively. Then the verifiers of the claimed link $A \leftrightarrow B$ are those nodes that are present in both the expected neighbor list of A and the expected neighbor list of B. Since each expected neighbor of A and B knows the expected neighbor lists of A and B, each can determine the verifiers of the claimed link $A \leftrightarrow B$. All the verifiers may not be within the communication range of both A and B. We will take this into account before each verifier provides a final response for the claimed link $A \leftrightarrow B$.

We now describe the Neighbor Verification Phase. Throughout this phase, each node explicitly announces the destination to which it is sending a packet. Also, each node can transmit a particular packet only once to each destination during a single round of this phase. Since the wireless medium is inherently prone to packet losses, this phase can be repeated for a number of rounds for those links over which verification packets were lost. Note that the round is repeated in its entirety rather than individual messages from the round.

Each node checks whether each of its links has at least k verifiers. If there does not exist at least k verifiers for a link, the link is dropped. Every verifier of a link also performs this operation. Let N_1 and N_2 be two expected neighboring nodes with at least k verifiers. N_1 initiates the link verification process by sending an authenticated packet to N_2 and explicitly announcing the address of N_2. N_1 waits until a timeout to receive an authenticated reply from N_2. When this communication happens between N_1 and N_2, no other node within the communication range of N_1, N_2 and the verifiers of the link N_1 and N_2 should transmit. A similar operation is performed for the link $N_2 \rightarrow N_1$.

The verifiers that hear the verification packet from N_1 can next hear one of three kinds of packets: a reply from N_2 to N_1 or the same packet from N_1 being relayed or some arbitrary packet being sent. Some legitimate verifiers might not actually hear either the transmission from N_1 or the transmission from N_2 or both. This is because the list of verifiers is obtained from the expected neighbor lists of N_1 and N_2 and not their actual neighbor list. Therefore, some verifiers may not be within the communication range of N_1 or N_2 or both. These verifiers mark themselves as *Dropped verifier* for that particular link. If a legitimate verifier hears the same packet from N_1 being relayed, it marks *Packet Relayed* for the link $N_1 \rightarrow N_2$. If the next packet that a legitimate verifier hears after hearing the verification packet from N_1, is a reply from N_2, it marks *Link Correct* for $N_1 \rightarrow N_2$. If a legitimate verifier hears some arbitrary packet being sent before the reply from N_2 comes, it does not mark anything at this time. If N_1 hears some arbitrary packet being sent before the reply from N_2 comes, it will repeat the phase for that link. Similar actions are taken when N_2 sends its verification packet to N_1. The phase will be repeated at most a predefined number of times in order to reduce the number of links that get dropped because of collisions and link errors. If at the end of these repetitions, a legitimate verifier has not marked anything for a particular link, it marks itself as *Dropped verifier* for that link. If either N_1 or N_2 have not marked *Link Correct* for $N_1 \leftrightarrow N_2$, the link is dropped.

The Neighbor Verification Phase is summarized in Table 3.

Table 3
The Neighbor Verification Phase.

1. S: Determine verifiers, $V_{S \leftrightarrow T}, \forall T \in \widetilde{N}(S)$
2. S: $\forall T, U \in \widetilde{N}(S)$, if $T \in \widetilde{N}(U)$ and $U \in \widetilde{N}(T)$, $S \in V_{T \leftrightarrow U}$
3. $S \rightarrow T$: $K_{S,T}$(Nonce N) $\forall T \in \widetilde{N}(S)$
4. $V_{S \leftrightarrow T}$: Hear whether the same packet is relayed to T
   If yes, mark *Packet Relayed*
   Else, donot mark anything at this point
5. $T \rightarrow S$: $K_{S,T}(h(N))$
6. $V_{S \leftrightarrow T}$: Hear whether the same packet is relayed to S
   If yes and if S was heard in Step 4, mark *Packet Relayed*
   Else if S was heard and the packet sent by T is not heard or if T is heard and S was not heard in Step 4, mark *Dropped Verifier*. Else, donot mark anything at this point
7. $V_{S \leftrightarrow T}$: If *Dropped Verifier* has been marked in Step 6, mark *Dropped Verifier*
   Else, if *Packet Relayed* has been marked in Step 4 and *Dropped Verifier* has not been marked in Step 6, or if *Packet Relayed* has been marked in Step 6, mark *Packet Relayed*
   Else, if after hearing S, the next packet heard was the reply sent by T, and only these two packets are heard during the communication between S and T, mark *Link Correct*
   Else, donot mark anything at this point
8. If S or T have not marked anything for $S \leftrightarrow T$, repeat the phase for this link

3.3. The response algorithm

After the Neighbor Verification Phase, each node would have either marked *Dropped Verifier* or *Link Correct* or *Packet Relayed* for every link for which it is a verifier.

A verifier, V, that has marked *Link Correct* or *Packet Relayed* for a link $A \leftrightarrow B$ during the Neighbor Verification phase, now determines whether it has marked *Link Correct* for the links $V \leftrightarrow A$ and $V \leftrightarrow B$. If V has marked anything else for these two links, then it changes its response to *Dropped Verifier* for $A \leftrightarrow B$. It is possible for both A and V to not be within communication range of each other and for V to mark $V \leftrightarrow A$ as *Link Correct*. This is possible if there is a malicious node or a chain of malicious nodes between V and A that had relayed the neighbor verification packets between V and A and V did not overhear the relay either because of collisions or link errors. Therefore the response of a legitimate verifier for a link may be incorrect. This is why we consider the response of multiple verifiers. Our simulations suggest that SEDINE performs well even in the presence of such collisions and link errors.

For each link $A \leftrightarrow B$, A, B, and the verifiers of the link $A \leftrightarrow B$ communicate their response for that link to each of the expected neighbors of A and B. Now A, B and each expected neighbor of A and B determines that the link $A \leftrightarrow B$ exists only if all of the following conditions hold:

(1) Both nodes claim that their link is correct.
(2) After removing verifiers that have marked themselves as *Dropped Verifier*, there still exists at least k verifiers for that link.
(3) Out of the k verifiers, there exists less than γ verifiers that have marked *Packet Relayed* for that link.

3.4. Incremental deployment of nodes

3.4.1. Assumptions

Our attack model is the same as that of the static network model. For the system model, we make the additional assumption that any node can distinguish between an incrementally deployed node and a node that was already present before incremental deployment. ID based authentication protocols, for instance, [6,2] can achieve this.

3.4.2. The protocol

With the additional assumption that we have made, the same neighbor discovery and neighbor verification protocol could be directly used for incrementally deployed nodes as well. When nodes are incrementally deployed, some nodes in the neighborhood of the newly deployed nodes would have been deployed much earlier and would have already built their neighbor lists while others would have been deployed along with the newly deployed nodes. Those nodes that have already built their neighbor lists only need to send these lists and verify that the link between them and the newly deployed nodes exist. Newly deployed nodes would build expected neighbor lists and would verify each and every link in order to build their first and second hop neighbor lists. To summarize, the protocol consists of the following steps.

(1) A newly deployed node will perform neighbor discovery and neighbor verification as described in Section 3.
(2) An already existing node that is present in the expected neighbor list of the newly deployed node will broadcast the expected neighbor list of the newly deployed node to its neighbors and will send the neighbor lists of each of its neighbors to the newly deployed node.
(3) An already existing node that is two hops away from the newly deployed node will send its neighbor list to the newly deployed node.
(4) An already existing node that is present in the expected neighbor list of the newly deployed node will perform neighbor verification in order to determine if the link between them is legitimate.

4. Analysis

4.1. Security analysis

We start by defining certain terms and notations.

Malicious Path. A malicious path between two nodes is a path that consists solely of malicious nodes, except possibly, the two end-points.

False Verifier. A false verifier of a link between two nodes claiming to be neighbors, is a node that is present in the expected neighbor list of both the nodes but is not within the communication range of at least one of the nodes.

True Verifier. A true verifier of a link between two nodes claiming to be neighbors, is a node that is within the communication range of both the nodes.

We now define the following notations.

- $\widetilde{N}(X)$: expected neighbor list of X
- $N(X)$: actual neighbor list of X
- L: set of all legitimate nodes in the network.
- M: set of all malicious nodes in the network.
- $R(A,B,V)$: response of verifier V for the link $A \rightarrow B$. We define the values taken by $R(A,B,V)$ as follows.

$$R(A,B,V) = \begin{cases} 0, & V \text{ marks } \textit{Dropped Verifier} \\ 1, & V \text{ marks } \textit{Link Correct} \\ -1, & V \text{ marks } \textit{Packet Relayed} \end{cases}$$

- $Y(A,B,X,p)$: we define $Y(A,B,X,p)$ as follows.

$$Y(A,B,X,p) = \begin{cases} 1, & X \in M \text{ relays packet } p \text{ sent by } A \text{ to } B \\ 0, & \text{otherwise} \end{cases}$$

- $H(A,B,V,p)$: the number of times $V \ (\in \widetilde{N}(A) \cap \widetilde{N}(B))$ hears packet p sent from A to B.
- $C(A,B)$: we define $C(A,B)$ as follows.

$$C(A,B) = \begin{cases} 1, & A \text{ and } B \text{ are within communication range of each other} \\ 0, & \text{otherwise} \end{cases}$$

- $P_M(A,B)$: set of malicious nodes forming a malicious path between non-neighboring nodes A and B. Formally, $P_M(A,B) = \{\{M_1, M_2, \ldots, M_n\} : M_i \in M \quad \forall i$ and $C(A,M_1) = C(M_1,M_2) = \cdots = C(M_n,B) = 1\}$.
- $S_M(A,B)$: set of all malicious paths between A and B.
- $V(A,B)$: set of all verifiers of the link $A \leftrightarrow B$. $V(A,B) = \{X :$ for $A \in \widetilde{N}(B)$ and $B \in \widetilde{N}(A),\ X \in \widetilde{N}(A)$ and $X \in \widetilde{N}(B)\}$.
- $V_F(A,B)$: set of all false verifiers of the link $A \leftrightarrow B$. $V_F(A,B) = \{X : X \in V(A,B)$ and $C(A,X) = 0$ or $C(B,X) = 0$ or both$\}$.
- $V_T(A,B)$: set of all true verifiers of the link $A \leftrightarrow B$. $V_T(A,B) = \{X : X \in V(A,B)$ and $C(A,X) = C(B,X) = 1\}$.

Theorem 4.1. *SEDINE prevents two non-neighboring nodes, A and B, from believing that they are neighbors, in the absence of packet losses, if at least one of the following conditions hold:*

(1) Both A and B are legitimate nodes or
(2) At least one of A and B is a legitimate node and there are no Sybil attacks.

Proof. The proof follows from the following Lemmas. □

Lemma 4.2. *In the absence of packet relaying attacks, an adversary cannot fool two non-neighboring legitimate nodes into believing that they are neighbors.*

Proof. Consider any two non-neighboring legitimate nodes, L_1 and L_2. In order for these two nodes to believe that they are neighbors, there must be a malicious path connecting L_1 and L_2 over which the adversary needs to relay the neighbor discovery and verification packets sent by L_1 to L_2, and vice versa. Therefore, in the absence of packet relaying attacks, an adversary cannot fool two non-neighboring legitimate nodes into believing that they are neighbors. □

Remark. The implication of Lemma 4.2 is that two legitimate non-neighboring nodes cannot be fooled into believing that they are neighbors in the absence of packet relaying attacks. While other attacks could assist this attack, they alone are not sufficient to make two non-neighboring legitimate nodes believe that they are neighbors. *Thus, guaranteeing security against packet relaying attacks is equivalent to guaranteeing security against all attacks for the problem of preventing two non-neighboring legitimate nodes from becoming neighbors.* The following results prove that SEDINE guarantees security against packet relaying attacks through overhearing. Thus, it follows that SEDINE guarantees security against all attacks that try to fool two non-neighboring legitimate nodes into believing that they are neighbors.

Lemma 4.3. *In the absence of packet losses, SEDINE prevents two non-neighboring legitimate nodes from becoming neighbors.*

Proof. The idea of the proof is simple. Consider any two non-neighboring legitimate nodes, L_1 and L_2, that have a malicious path connecting them. Assume that after the Neighbor Discovery Phase, they have been made to believe that they are neighbors. During the Neighbor Verification Phase, when L_1 sends a verification packet to L_2, in order for L_2 to receive this packet, it has to traverse through the malicious path between L_1 and L_2. For this to happen, the malicious nodes in the malicious path have to replay the verification packet sent by L_1. Since L_1 is a verifier of its own links, in the absence of packet losses and collisions, L_1 will hear the replay. Since the next packet that L_1 hears is not the reply from L_2, it will not accept that the link $L_1 \leftrightarrow L_2$ exists.

We will now formalize this proof based on the framework that we developed.

Let $L_1 \in L$ and $L_2 \in L$ s.t. $C(L_1,L_2) = 0$.

If $|S_M(L_1,L_2)| = 0$, then $L_1 \notin \widetilde{N}(L_2)$ and $L_2 \notin \widetilde{N}(L_1)$. Therefore, $L_1 \notin N(L_2)$ and $L_2 \notin N(L_1)$.

Let $|S_M(L_1,L_2)| > 0$ and let $L_1 \in \widetilde{N}(L_2)$ and $L_2 \in \widetilde{N}(L_1)$ after the Neighbor Discovery Phase.

We have the following cases.

Case 1: $|S_M(L_1,L_2)| = 1$.

There are now two possibilities.

First, let $|P_M(L_1,L_2)| = 1$. This means that L_1 and L_2 are within two hops of each other and there exists $M_1 \in M$ such that $P_M(L_1,L_2) = \{M_1\}$ and that there exists no other malicious path between L_1 and L_2.

During the Neighbor Verification Phase, let L_1 send a verification packet, p, to L_2. For L_2 to receive this packet, $Y(L_1,L_2,M_1,p) = 1$ since $C(L_1,L_2) = 0$.

But in this case, $H(L_1,L_2,L_1,p) > 1$, since by our assumptions L_1 can hear any packet sent by M_1. Therefore, $R(L_1,L_2,L_1) = -1$. Since L_1 is the source and it detects packet relaying, it drops the link. Therefore, $L_2 \notin N(L_1)$.

Now, consider the other case in which $|P_M(L_1,L_2)| > 1$. This means that there exists a chain of malicious nodes $M_1, M_2, \ldots, M_i \in M$ s.t. $P_M(L_1,L_2) = \{M_1, M_2, \ldots, M_i\}$, $i > 1$ and that there exists no other malicious path between L_1 and L_2.

Here, $C(L_1,M_1) = 1$ and $C(M_i,L_2) = 1$. During the Neighbor Verification Phase, let L_1 send a verification packet, p, to L_2. For L_2 to receive p, $Y(L_1,L_2,M_i,p) = 1$. But for M_i to receive p, either $Y(L_1,L_2,M_{i-1},p) = 1$ or $Y(L_1,L_2,M_{i-1},E(p)) = 1$ where $E(p)$ is an encrypted version of packet p and M_i has the key to decrypt it. Applying this iteratively, for L_2 to receive packet p, either $Y(L_1,L_2,M_1,p) = 1$ or $Y(L_1,L_2,M_1,E(p)) = 1$.

If $Y(L_1,L_2,M_1,p) = 1, H(L_1,L_2,L_1,p) > 1$. Therefore, $R(L_1,L_2,L_1) = -1$. So L_1 drops the link. On the other hand, if $Y(L_1,L_2,M_1,E(p)) = 1$, then L_1 does not mark anything and repeats the Neighbor Verification Phase for $L_1 \leftrightarrow L_2$.

When the phase is repeated, the same procedure follows. If $Y(L_1,L_2,M_1,p) = 1$ during any of the phase repetitions, then $R(L_1,L_2,L_1) = -1$. On the other hand, if $Y(L_1,L_2,M_1,E(p)) = 1$ for every repetition of the Neighbor Verification Phase, then $R(L_1,L_2,L_1) = 0$. Since L_1 has to have $R(L_1,L_2,L_1) = 1$ for $L_1 \leftrightarrow L_2$ to be validated, L_1 drops the link.

Case 2: $|S_M(L_1,L_2)| > 1$.

This means that there exists multiple malicious paths between L_1 and L_2. Irrespective of the malicious path taken by the verification packet, $R(L_1,L_2,L_1) \neq 1$ by Case 1. Therefore, L_1 drops the link. Therefore, $L_2 \notin N(L_1)$ and $L_1 \notin N(L_2)$. □

Lemma 4.4. *SEDINE prevents two non-neighboring nodes, one legitimate and the other malicious, from becoming neighbors, in the absence of packet losses and Sybil attacks.*

Proof. Let $L_1 \in L$ and $M_1 \in M$ s.t. $C(L_1,M_1) = 0$.

The proof is similar to the proof of Lemma 4.3, except that in the absence of Sybil attacks (the effects of which are described at the end of this proof), only the legitimate node will now mark that the link between itself and the malicious node does not exist. The malicious node might or might not mark that the link does not exist.

But since we assume that the links are bi-directional, it is enough for one node to claim that the link does not exist, in order to drop the link.

Thus, SEDINE prevents two non-neighboring nodes from becoming neighbors even when one of them is malicious. Therefore, $M_1 \notin N(L_1)$ and so M_1 cannot convince any of its actual neighbors that $L_1 \in N(M_1)$. □

We briefly explain the significance of Sybil attacks in Lemma 4.4. Let $A \in L$ and $M_1 \in M$ such that $C(A,M_1) = 0$ but there exists $M_2 \in M$ such that $C(A,M_2) = C(M_2,M_1) = 1$. M_1 and M_2 can collude and exchange identities. So, when A tries to find its neighbors, M_2 would pose both as M_2 and M_1 and fool A into believing that both M_2 and M_1 are its neighbors. However, here A is only adding a node that is within its communication range but possessing multiple identities. Newsome et al. [18,21,25] suggest approaches to detect nodes possessing multiple identities in sensor and mobile ad-hoc networks. However, protecting against Sybil attacks is still an open problem. It is important to note that *Sybil attacks cannot fool two non-neighboring legitimate nodes into believing that they are neighbors*. The correctness of SEDINE is affected by the Sybil attack when a malicious node takes multiple identities and creates a spurious link between a legitimate node and a node with one of the false identities.

From the above lemmas, Theorem 4.1 directly follows.

Theorem 4.1 guarantees that irrespective of any security attack launched against SEDINE other than those mentioned in our assumptions, two legitimate non-neighboring nodes cannot be fooled to become neighbors, in the absence of packet losses and collisions. In Section 5, we will provide a number of examples to understand this result better.

In addition, it can be observed from the proof of Theorem 4.1 that *in the absence of packet losses and collisions, in order to guarantee that two non-neighboring legitimate nodes do not get fooled into believing that they are neighbors, the decision taken by the corresponding two legitimate nodes is sufficient. The decisions taken by the other verifiers or even the availability of other verifiers do not matter*. This implies that when packet losses are negligible and the network is so sparsely distributed that collisions are negligible, it is easy for two legitimate non-neighboring nodes to ensure that they do not believe that they are neighbors. The same results hold when Sybil attacks are absent and one of the nodes is legitimate and the other malicious.

Note that the results in Theorem 4.1 do not hold in the presence of packet losses. Therefore, though Theorem 4.1 suggests that verifiers are not necessary to determine whether a link is legitimate, this is only true in the absence of packet losses, and hence we collect decisions from multiple verifiers (in Section 3) in order to account for packet losses. We show through simulations that SEDINE performs well even in the presence of packet losses and collisions.

Corollary 4.5 (*Corollary to Theorem 4.1*). *In the absence of packet losses and Sybil attacks, SEDINE guarantees:*

1. *The neighbors of a legitimate node, S, will only be those nodes that are within the one-hop communication range of S. The legitimate nodes that are neighbors of a malicious node, X, will only be those nodes that are within the one-hop communication range of X.*
2. *For a legitimate node, S, let $T \in \widetilde{N}(S)$ and $V \in \widetilde{N}(T)$. If either of T or V are legitimate, S will accept the link $T \leftrightarrow V$ to exist, only if V is within the one-hop communication range of T.*

Thus, apart from securely determining its one-hop neighbors, a node can also verify the neighbors of each of its neighbors using SEDINE.

4.2. Connectivity analysis

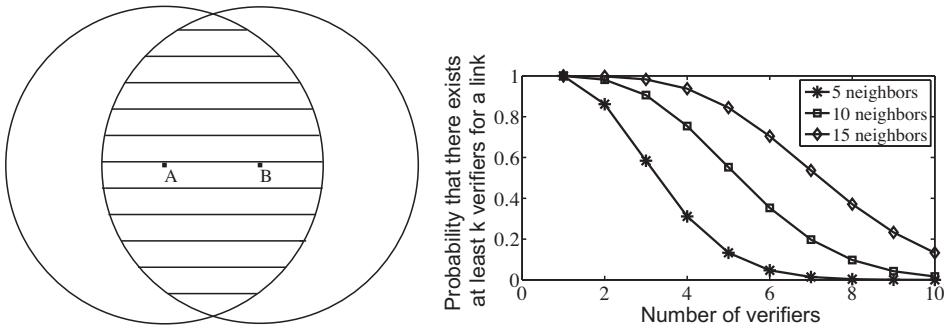
We now analyze how the protocol performs in the absence of attacks, in terms of the number of legitimate links dropped because of the absence of k verifiers, so that we can empirically find a good value of k that will provide the desired connectivity required by the application.

We abstract the communication range of each node by a circle of radius r. Let two neighboring nodes, A and B, be separated by a distance d. Then, the verifiers of $A \leftrightarrow B$ are those nodes that are present in the shaded region in Fig. 2(a). The area of this shaded region is given by $Ar_{omni} = 2r^2 \cos^{-1}\left(\frac{d}{2r}\right) - \frac{d}{2}\sqrt{4r^2 - d^2}$. Ar_{omni} is minimum when B is on the edge of the communication range of A. This area is given by $Ar_{omni} = r^2\left(\frac{2\pi}{3} - \frac{\sqrt{3}}{2}\right)$.

Suppose there are a total of N nodes uniformly and randomly distributed in a field of area, Ar. For the link between A and B to exist, we need at least k verifiers in the shaded region in Fig. 2(a). The probability that at least k verifiers are present in this shaded area (Ar_{omni}) is given by $\sum_{i=k}^{N}\binom{N}{i}\left(\frac{Ar_{omni}}{Ar}\right)^i\left(1 - \frac{Ar_{omni}}{Ar}\right)^{N-i}$. Assuming 100 nodes and varying the average number of neighbors of a node between 5 and 15 (by varying the area in which the nodes are deployed) and with a communication range of 30 m, Fig. 2(b) shows the probability of having at least k verifiers in this minimum area, for varying k. Since the source of a link is a verifier, there always exists one verifier for a link when we use SEDINE. For a typical density of 10 neighbors a node, there exists at least two verifiers for a link with a probability >0.95, and there exists at least three verifiers for a link with a probability >0.9. Since this minimum area occurs when the two nodes are at the edge of each other's communication range, this probability is actually a lower bound.

5. Discussion

We will now consider a number of sample attacks against SEDINE and discuss how they are countered.



(a) At least k nodes need to be neighbors of both A and B

(b) Probability that there exists at least k verifiers for a link

Fig. 2. Connectivity analysis.

5.1. Attacks against the Neighbor Discovery Phase

5.1.1. Attacks against the discovery of expected neighbors

During this phase, the adversary can replay HELLO broadcasts and make two non-neighboring nodes add each other in their respective expected neighbor lists. However such attacks are countered during the Neighbor Verification Phase. The adversary cannot tamper with authenticity verification between two legitimate nodes during this stage since they share a key.

5.1.2. Attacks against determining verifiers

Each node, after finding its expected neighbors, finds the verifiers for each of its links using the procedure in Tables 2 and 3. Now, consider two non-neighboring legitimate nodes L_1 and L_2 and a malicious node M such that there is a path $L_1 \leftrightarrow M \leftrightarrow L_2$. M can fabricate a neighbor list for L_1 and L_2, encrypt them with randomly generated keys, broadcast them to L_2 and L_1 respectively and reveal the keys that it generated later. However, since M does not possess $h(N_{L_2,L_1})$ and $h(N_{L_1,L_2})$, L_2 and L_1 will immediately detect the malicious activity and not accept the neighbor lists.

Now suppose that one of the non-neighboring nodes is malicious. For instance, let us replace L_2 by M_2 which is malicious. If M_2 colludes with M, then M can fabricate the expected neighbor list of M_2. Then L_1 would form a wrong verifier list for the claimed link $L_1 \leftrightarrow M_2$. However, this is not enough for the adversary to make L_1 and M_2 believe that they are neighbors.

Remember that in the absence of Sybil attacks, packet losses and collisions, we can observe from the proof of Theorem 4.1 that only the decision of L_1 is required in order to identify whether a link exists. This is because L_1 is a verifier of its own links (and this fact cannot be manipulated by a malicious node) and therefore it would overhear M's relay to M_2 during the Neighbor Verification Phase.

Even in the presence of packet losses or collisions, in order for the adversary to ensure a successful attack, the neighbor list fabricated by M must contain at least k verifiers out of which at least $k - \gamma$ must be colluding compromised malicious nodes and L_1 must be prevented from overhearing M's relay. From the point of view of the adversary, any legitimate node in this list has the potential to become a *Dropped Verifier* or could report*Packet Relayed* during the Neighbor Verification Phase. Since at least k verifiers are required for each link to be verified and less than γ verifiers can report*Packet Relayed*, the adversary requires at least $k - \gamma$ compromised and colluding malicious nodes to launch this attack. This is a sophisticated attack and has to occur during the Neighbor Discovery Phase.

5.2. Attacks against the Neighbor Verification Phase

During this stage, the adversary can tamper with the verification packet, launch wormhole relaying attacks or encrypt verification packets and launch wormhole attacks. The proof of Theorem 4.1 clearly shows that SEDINE is resilient to the latter two attacks. Tampering with the verification packet cannot affect SEDINE since the source node and the destination will both detect this attack as the packet has been encrypted with the secret key shared between them.

5.3. Out-of-band channel and directional antenna attacks

We now briefly describe how to handle out-of-band channel and attacks using high powered transmission (including directional antennas). For a directional antenna attack, one malicious node is sufficient. SEDINE can account for the directional antenna attack if there are verifiers in the direction of relay. Depending on the number of orientations of the directional antenna, it may be unlikely that a verifier is present in that direction. Therefore, SEDINE, by itself, cannot provide any provable guarantees against this attack.

In order to launch an out-of-band channel attack, at least two malicious nodes with out-of-band channels are required. We now show how the technique in [17] can be used in conjunction with SEDINE in order to further secure neighbor discovery. In [17], Maheshwari et al. use connectivity information to form connectivity maps at each node. Using these maps, the authors propose an algorithm to detect false links in the network. This algorithm requires that the two non-neighboring nodes are located sufficiently far so that their communication ranges do not intersect. Otherwise, it may not work. Since both out-of-band channel attacks and attacks using

high powered transmission are most useful to the adversary when the wormhole is sufficiently long, this technique can be used to mitigate such attacks. However, *the criticality of securing neighbor discovery still lies in the fact that the adversary does not need these powerful attacks to disrupt neighbor discovery, and therefore it is important to provide security guarantees even in the absence of these attacks.*

5.4. Sybil and jamming attacks

As we saw in Theorem 4.1, Sybil attacks cannot make two non-neighboring legitimate nodes believe that they are neighbors. However, they can make a malicious node, and a non-neighboring legitimate node believe that they are neighbors. In order to do this, the malicious node must first collude with another malicious node. Also, the malicious nodes must be located such that one of them is a neighbor of both the legitimate node and the other malicious node.

Jamming attacks are generally easy to launch. However, in the absence of packet losses and collisions, they cannot disrupt the functioning of SEDINE. In order to launch a jamming attack against SEDINE, at least two colluding malicious nodes are required. Not only must these malicious nodes be positioned appropriately, but also they have to synchronize with each other. The jammer must be placed such that it only jams all the verifiers, and not the receiver. However, even if the adversary successfully manages to achieve this, during the verification phase, SEDINE will accept a link to be valid only if the next message received is a response to the verification message. Therefore, jamming attacks cannot make two non-neighboring nodes believe that they are neighbors. They can however be used to disconnect the network. When this happens, one can use techniques to detect the presence of jammers, and report them to a base station.

6. Simulations

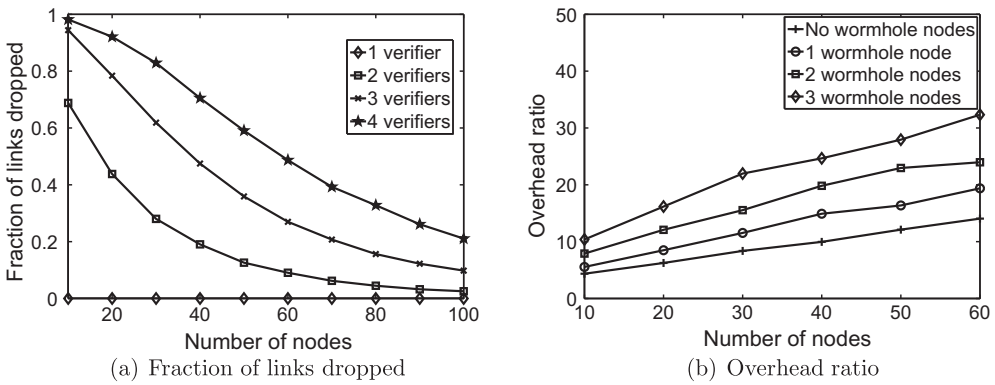
We have performed three numerical experiments to quantify the performance of SEDINE. We explain them in this section.

6.1. Fraction of links dropped

This experiment identifies the effect of topology on the fraction of links dropped by SEDINE and compares it with that of the directional antenna protocol [7]. In order to perform this comparison, we use the same settings as used in [7]. The purpose of this experiment is also to determine a suitable value for k. This experiment is performed using MATLAB. Nodes are uniformly and randomly deployed in a 100×100 square field. The number of nodes in the field varies from 10 to 100.

Here, the fraction of legitimate links that get dropped due to the absence of k verifiers for those links, is simulated for different k. As assumed in [7], this simulation is done without malicious nodes. The simulation is run 1000 times and the results are averaged. Since, in our protocol, each node, itself, is a verifier of the link that it is a part of, no links get dropped when $k = 1$. This can be seen in Fig. 3(a). For $k > 1$, the fraction of links dropped decreases as the node density increases since the probability that k verifiers exist for a link increases as the node density increases. For a typical neighborhood density of 10 neighbors a node with an omni-directional antenna (corresponding to approximately 33 neighbors with a directional antenna [7]), the strict neighbor discovery protocol of the directional antenna approach, with one verifier, drops 40% of the legitimate links [7] while SEDINE does not drop any links with one verifier. Further, comparing the same number of neighbors (33 for both SEDINE and the directional antenna protocol), SEDINE drops less than 4% of the legitimate links even with two verifiers.

For a given network, the value of k should be chosen based on the network density and the level of security that is required by the application. For example, if we have a circular topology (with total number of nodes > 3) in which each node has exactly two neighbors, then we cannot choose $k > 1$ (since, in this case, all legitimate links would be dropped). However, since each node is itself a verifier of each of its links, choosing $k = 1$ still prevents two non-neighboring nodes, at least one of which is legitimate, from believing that they are neighbors, in the absence of collisions and link errors. In a network where nodes are randomly deployed, choosing a large value for k will result in the exclusion of nodes with few neighbors. *In applications where security is so critical that dropping of legitimate*



(a) Fraction of links dropped (b) Overhead ratio

Fig. 3. Experimental results for fraction of links dropped and message overhead.

links is not as crucial, for any network density, SEDINE guarantees that two legitimate non-neighboring nodes cannot become neighbors, in the absence of packet losses (Theorem 4.1).

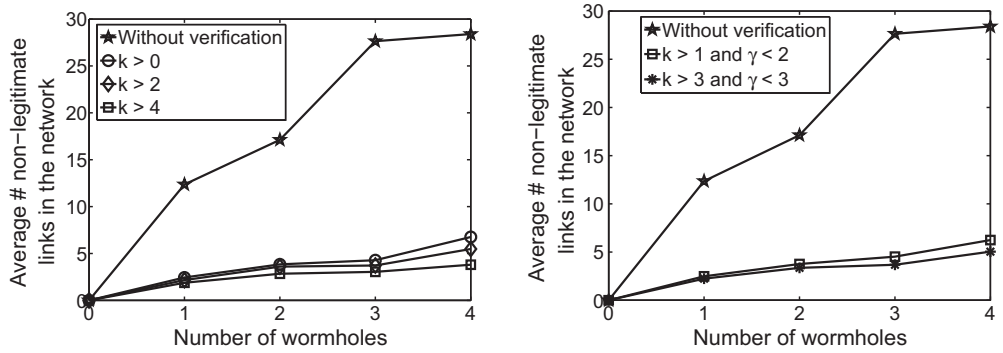
6.2. Communication overhead with and without verification

We now compare the communication overhead when verification is used to the scenario in which verification is not used. The metric used here is the ratio of the total number of messages sent and received by a node, on average, using SEDINE and using an insecure protocol (that consists only of a HELLO message and a reply to the HELLO packet). We call this metric the "Overhead Ratio". The total number of messages sent and received by a node using SEDINE is the total number of messages sent and received in both the Neighbor Discovery and the Neighbor Verification Phases. Jist SWANS [1] has been used for this simulation. The IEEE 802.11 MAC protocol has been used. Nodes are deployed uniformly and randomly in a 100×100 m^2 field. The number of legitimate nodes is varied from 10 to 60, and the number of malicious nodes is varied from 0 to 3. The simulation is run 100 times and the results are averaged.

From Fig. 3(b), we observe that when we take overhearing into account, the communication overhead ratio increases linearly with the total number of nodes. This can be understood as follows. If a particular node has N neighbors and each of these neighbors has $O(N)$ neighbors, then the protocol without verification would only require $O(N)$ messages to be exchanged whereas SEDINE would require $O(N^2)$ messages to be exchanged since each node overhears every packet sent by each of its neighbors as well. We also see that as the number of malicious nodes increases, the communication overhead ratio also increases. This is because of a significant increase in the overheard traffic caused by the packet relaying attack.

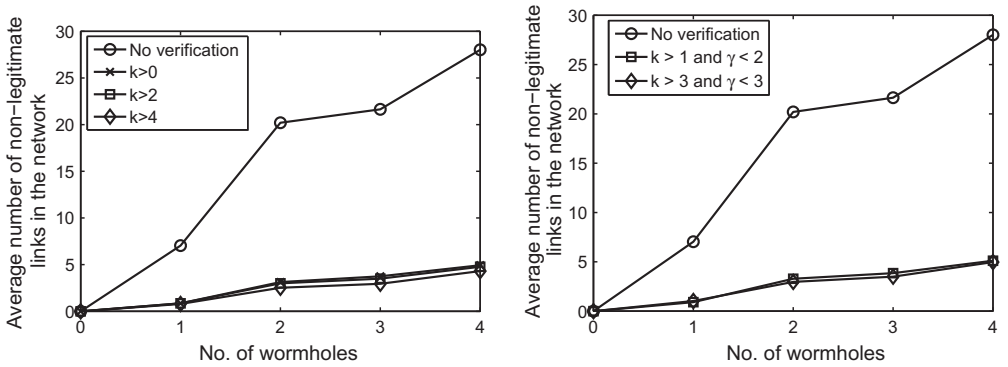
6.3. Number of non-legitimate links with \without verification

In this experiment, we show the advantage of using verification. We also consider the effect of packet losses due to collisions and link errors. This experiment has been simulated using Jist SWANS. The IEEE 802.11 MAC protocol has been used. The simulation is run 25 times and the results are averaged. We consider the worst-case scenario in which malicious nodes relay whatever packet they hear. 50 legitimate nodes are uniformly and randomly deployed in a 100×100 m^2 field. The number of malicious nodes is varied from 0 to 4 and the communication range is roughly 32 m. Fig. 4 shows the results when there are only collisions, and no packet losses. Fig. 5 shows the corresponding results when there are both packet losses and collisions. The packet losses in these results are independent over time and across nodes. The probability of a packet loss over



(a) With $\gamma = 0$ (b) With $\gamma \leq k/2$

Fig. 4. Only collisions.



(a) With $\gamma = 0$ (b) With $\gamma \leq k/2$

Fig. 5. Both collisions and packet losses.

a link is a random number between 0 and 0.3. In order to account for packet losses, we allow the protocol to be rerun multiple times.

Let k represent the minimum number of verifiers required to validate a link and let γ represent the maximum number of verifiers that can report *Packet Relayed* for a link so that the link still gets validated. By increasing γ, we can prevent compromised malicious nodes from framing legitimate links. However, increasing γ decreases the chance of detecting a non-legitimate link. From Figs. 4(a) and (b), 5(a) and (b), we see that for the same k, increase in γ results in a slight increase in the number of non-legitimate links in the network.

We observe that the total number of non-legitimate links in the network decreases as the number of verifiers increases. There is a trade-off here since arbitrarily increasing the number of verifiers results in an increase in the number of legitimate links being dropped. We also observe that even as we increase the number of malicious nodes, the corresponding rate of increase of the number of non-legitimate links is much lower when verification is used than when verification is not used. Further, these results are true whether there are packet losses or not. The reason that the results are similar is that we rerun the protocol multiple times when there are packet losses.

We also observe that increasing the number of verifiers does not result in a significant decrease in the number of non-legitimate links. However, there is a huge decrease in the number of non-legitimate links when even one verifier is used compared to when there is no verification. Since using one verifier does not result in any legitimate links being dropped (Fig. 3(a)), there is a lot of incentive to use verification.

7. Conclusion

Securing the neighbor discovery protocol is a critical problem in wireless ad-hoc and sensor networks. SEDINE not only tries to prevent legitimate non-neighboring nodes from being fooled to believe that they are neighbors, but also malicious nodes from becoming neighbors to legitimate non-neighboring nodes. Further, SEDINE supports incremental node deployment. Our simulation results show that SEDINE is successful in preventing a huge fraction of non-legitimate links from being formed in a lossy wireless communication environment. Some of the open issues include providing provable security guarantees for out-of-band channel, power controlled, and Sybil attacks, studying the effects of denial of service attacks against neighbor discovery, and designing secure neighbor discovery protocols for mobile networks.

References

[1] Rimon Barr. Swans user guide, 2004.
[2] Haowen Chan, A. Perrig, Pike: Peer intermediaries for key establishment in sensor networks, in: IEEE INFOCOM 2005, 2005.
[3] Haowen Chan, A. Perrig, D. Song, Random key predistribution schemes for sensor networks, in: Symposium on Security and Privacy, 2003.
[4] W. Du, J. Deng, Y. Han, P. Varshney, A pair-wise key pre-distribution scheme for wireless sensor networks, in: ACM CCS, 2003.
[5] J. Eriksson, S.V. Krishnamurthy, M. Faloutsos, Truelink: a practical countermeasure to the wormhole attack in wireless networks, ICNP (2006).
[6] L. Gong, D.J. Wheeler, A matrix key-distribution scheme, Journal of Cryptology (1990).
[7] L. Hu, D. Evans, Using directional antennas to prevent wormhole attacks, in: Network and Distributed System Security Symposium, 2004.
[8] Y.C. Hu, A. Perrig, D.B. Johnson, Rushing attacks and defense in wireless ad hoc network routing protocols, in: ACM WiSe Workshop, 2003.
[9] D. Johnson, D. Maltz, J. Broch, The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, in: Ad Hoc Networking, Addison-Weasley, 2001.
[10] I. Khalil, S. Bagchi, N.B. Shroff, Liteworp: A lightweight countermeasure for the wormhole attack in multihop wireless networks, in: DSN, 2005.
[11] R. Poovendran L. Lazos, S. Capkun, Rope: Robust position estimation in wireless sensor networks, in: IPSN, 2005.
[12] Suk-Bok Lee, Yoon-Hwa Choi, A secure alternate path routing in sensor networks, Computer Communications (2006).
[13] D. Liu, P. Ning, Establishing pair-wise keys in distributed sensor networks, in: ACM CCS, 2003.
[14] D. Liu, P. Ning, W. Du, Detecting malicious beacon nodes for secure location discovery in wireless sensor networks, in: ICDCS, 2005.
[15] Donggang Liu, Peng Ning, W.K. Du, Attack-resistant location estimation in sensor networks, in: IPSN, 2005.
[16] Hong Luo, Zengjun Zhang, Yonghe Liu, Recoda: reliable forwarding of correlated data in sensor networks with low latency, in: IWCMC, 2007.
[17] R. Maheshwari, Jie Gao, S.R. Das, Detecting wormhole attacks in wireless networks using connectivity information, in: IEEE INFOCOM, 2007.
[18] J. Newsome, E. Shi, D. Song, A. Perrig, The sybil attack in sensor networks: Analysis and defenses, in: IEEE/ACM IPSN, 2004.
[19] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Capkun, J-P. Hubaux, Secure neighborhood discovery: a fundamental element for mobile ad hoc networking, IEEE Communications Magazine (2008).
[20] C.E. Perkins, E.M. Royer, Ad-hoc on-demand distance vector routing, in: IEEE WMCSA, 1999.
[21] Chris Piro, Clay Shields, Brian Neil Levine. Detecting the sybil attack in ad hoc networks, in: Proc. IEEE SecureComm, 2006.
[22] Marcin Poturalski, Panos Papadimitratos, Jean-Pierre Hubaux. Secure neighbor discovery in wireless networks: formal investigation of possibility, in: ASIACCS, 2008.
[23] Kasper Bonne Rasmussen, Srdjan Capkun, Implications of radio fingerprinting on the security of sensor networks, in: SecureComm, 2007.
[24] W. Yuan, S.V. Krishnamurthy, S.K. Tripathi, Improving the reliability of event reports in wireless sensor networks, in: ISCC, 2004.
[25] Qinghua Zhang, Pan Wang, D.S. Reeves, Peng Ning, Defending against Sybil attacks in sensor networks, in: 25th IEEE International Conference on Distributed Computing Systems Workshops, 2005.
[26] Y. Zhang, W. Liu, W. Lou, Y. Fang, Location-based compromise-tolerant security mechanisms for wireless sensor networks, IEEE Journal on Selected Areas in Communications (2006).

Srikanth Hariharan received his B.Tech. degree from the Indian Institute of Technology, Madras, in 2006, and his M.S. degree from Purdue University, West Lafayette, IN, in 2007. He is currently a Doctoral candidate in the Department of Electrical and Computer Engineering at the Ohio State University. His research interests include data aggregation in wireless sensor networks, target tracking, multi-hop wireless scheduling, and wireless security.

Ness B. Shroff (F'07) received his Ph.D. degree from Columbia University, NY, in 1994 and joined Purdue University 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, a chaired Professor of ECE and CSE. He is also a guest chaired professor of Wireless Communications and Networking in the department of Electronic Engineering at Tsinghua University. 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.

Shroff is a past editor for IEEE/ACM Trans. on Networking and the IEEE Communications Letters and current editor of the Computer Networks Journal. He has served as the technical program co-chair and general co-chair of several major conferences and workshops, such as the IEEE INFOCOM 2003, ACM Mobihoc 2008, IEEE CCW 1999, and WICON 2008. He was also a co-organizer of the NSF workshop on Fundamental Research in Networking (2003) and the NSF Workshop on the Future of Wireless Networks (2009). Shroff is a fellow of the IEEE. He received the IEEE INFOCOM 2008 best paper award, 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).

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.