Secure Neighbor Discovery through Overhearing in Static Multihop Wireless Networks

Srikanth Hariharan
Department of ECE
The Ohio State University
Columbus, OH - 43210
Email: harihars@ece.osu.edu

Ness B. Shroff
Departments of ECE and CSE
The Ohio State University
Columbus, OH - 43210
Email: shroff@ece.osu.edu

Saurabh Bagchi
School of ECE
Purdue University
West Lafayette, IN - 47906
Email: sbagchi@purdue.edu

***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.**

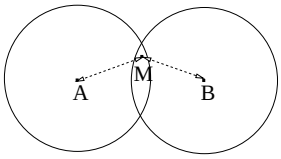
I. Introduction

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 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.

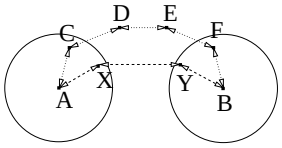
An adversary intending to disrupt the neighbor discovery protocol, will try to make two non-neighboring nodes believe that they are neighbors or will prevent two neighboring nodes from becoming neighbors. By launching the former attack, the adversary can in turn attack protocols that need accurate neighbor information. For example, an adversary can attack routing protocols such as AODV [1] and DSR [2] 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.

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

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 (Figure 1(a)). If the neighbor discovery protocol simply consists of broadcasting a HELLO packet and receiving a response for the HELLO packet (as is typical [1], [2], [3]), 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 (Figure 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 $A - X - Y - B$.



(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

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 Figure 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 [3], [4]. This is the key motivation for working on this problem.

We now overview related work. Typically, most work (e.g., [3]) 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 [5], [6]. 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. [7] show that timing information alone cannot guarantee secure neighbor discovery.

Another class of protocols rely on specialized hardware such as directional antennas [4] or advanced physical layer features [8]. The directional antenna protocol [4] substantially degrades network connectivity and does not consider framing attacks. [8] 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.

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., [9] have also explained the importance of providing security guarantees for neighbor discovery in their survey paper. Another interesting paper by Maheshwari et. al., [10] 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. The main contributions of this paper are as follows:

- We develop a *provably secure* neighbor discovery protocol that *does not require any specialized hardware, 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*.
- 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.

The rest of this paper is organized as follows: We present the system model and detail our assumptions in Section II. Section III describes the neighbor discovery protocol for static multi-hop wireless networks. In Section IV, we provide security analysis for SEDINE. In Section V, we present our simulation results. Finally, we conclude our paper and discuss open problems in Section VI.

II. System Model and Assumptions

A. System and Attack 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 [11], [12], [13] that will allow any two nodes to establish a secure communication channel between them). In our model, we allow packet losses to occur due to link errors or collisions. 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 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 simple neighbor discovery mechanism that cannot provide any 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. We allow malicious nodes, both compromised and external, to 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 launching a wormhole attack in the future. *We do not consider denial of service attacks that prevent two neighboring nodes from becoming neighbors, physical layer jamming attacks, and physical destruction of nodes.*

III. 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 some 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.

A. The Neighbor Discovery Phase

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 [11], [12], [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.

2) Determining the expected second hop neighbors: Once a 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 the claimed links.

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 I, 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.

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 I. If some 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.

The Neighbor Discovery Phase is summarized in Table I:

TABLE I
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: $K'_{S,Bcast}$, $K_{T,Bcast}\ \forall\ T \in \widetilde{N}(S)$.

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

B. 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 II.

TABLE II
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, don't 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, don't 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, don't mark anything at this point.
8.	If S or T have not marked anything for $S \leftrightarrow T$, repeat the phase for this link.

C. 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.

IV. SECURITY ANALYSIS

Before we begin the analysis, we first define a *malicious path* between two nodes as a path that consists solely of malicious nodes, except possibly the two end-points.

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: SEDINE prevents two non-neighboring legitimate nodes from becoming neighbors, in the absence of packet losses.

Proof: We present the idea of the proof here. The details can be found in [14]. 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. In order 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. ■

Lemma 4.3: 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: See proof in [14]. ■

We briefly explain the significance of Sybil attacks in Lemma 4.3. Let A be a legitimate node, and M_1 be a malicious node such that A and M_1 are not within communication range of each other, but there exists a malicious node M_2 that is within the communication range of both A and M_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. [15], [16], [17] 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 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 don't 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.

Corollary 4.4 (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 be in the expected neighbor list of S, and V be in the expected neighbor list of 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.

V. Simulations

In this section, we quantify the performance of SEDINE through numerical experiments.

A. 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 [4]. In order to perform this comparison, we use the same settings as used in [4]. 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 [4], 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 Figure 2. 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 [4]), the strict neighbor discovery protocol of the directional antenna approach, with one verifier, drops 40% of the legitimate links [4] 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. 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 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*).*

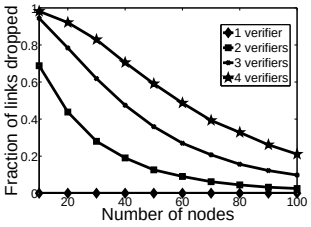


Fig. 2. Fraction of links dropped due to the necessity of the existence of k verifiers for every link

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

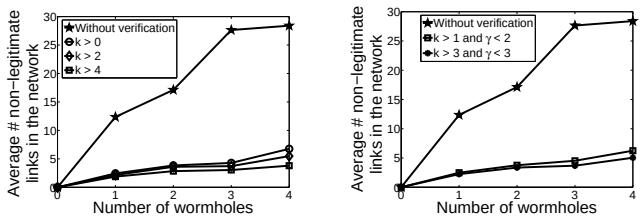
In this experiment, we show the advantage of using verification. We also consider the effect of packet losses due to collisions and link errors. This experiment has been simulated using Jist SWANS. The IEEE 802.11 MAC protocol has been used. The simulation is run 25 times and the results are averaged. 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.

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 Figures 3(b) and 3(a), 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.

VI. 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. 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.



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

Fig. 3. Number of non-legitimate links with and without verification

References

[1] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *IEEE WMCSA*, 1999.

[2] D. Johnson, D. Maltz, and J. Broch, "The dynamic source routing protocol for multihop wireless ad hoc networks," in *Ad Hoc Networking, Addison-Weasley*, 2001.

[3] I. Khalil, S. Bagchi, and N. B. Shroff, "Liteworp: A lightweight countermeasure for the wormhole attack in multihop wireless networks," in *DSN*, 2005.

[4] L. Hu and D. Evans, "Using directional antennas to prevent wormhole attacks," in *Network and Distributed System Security Symposium*, 2004.

[5] Y. C. Hu, A. Perrig, and D. B. Johnson, "Rushing attacks and defense in wireless ad hoc network routing protocols," in *ACM WiSe Workshop*, 2003.

[6] D. Liu, P. Ning, and W. Du, "Detecting malicious beacon nodes for secure location discovery in wireless sensor networks," in *ICDCS*, 2005.

[7] M. Poturalski, P. Papadimitratos, and J.-P. Hubaux, "Secure neighbor discovery in wireless networks: formal investigation of possibility," in *ASIACCS*, 2008.

[8] K. B. Rasmussen and S. Capkun, "Implications of radio fingerprinting on the security of sensor networks," in *SecureComm*, 2007.

[9] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Capkun, and J.-P. Hubaux, "Secure neighborhood discovery: A fundamental element for mobile ad hoc networking," *IEEE Communications Magazine*, 2008.

[10] R. Maheshwari, J. Gao, and S. Das, "Detecting wormhole attacks in wireless networks using connectivity information," in *INFOCOM*, 2007.

[11] H. Chan, A. Perrig, and D. Song, "Random key predistribution schemes for sensor networks," in *Symposium on Security and Privacy*, 2003.

[12] W. Du, J. Deng, Y. Han, and P. Varshney, "A pair-wise key predistribution scheme for wireless sensor networks," in *ACM CCS*, 2003.

[13] D. Liu and P. Ning, "Establishing pair-wise keys in distributed sensor networks," in *ACM CCS*, 2003.

[14] S. Hariharan, N. B. Shroff, and S. Bagchi, "Secure neighbor discovery in wireless sensor networks," Tech. Rep., 2007. [Online]. Available: http://www.ece.osu.edu/∼harihars/sedine.pdf

[15] J. Newsome, E. Shi, D. Song, and A. Perrig, "The sybil attack in sensor networks: Analysis and defenses," in *IEEE/ACM IPSN*, 2004.

[16] C. Piro, C. Shields, and B. N. Levine, "Detecting the Sybil Attack in Ad hoc Networks," in *Proc. IEEE SecureComm*, 2006.

[17] Q. Zhang, P. Wang, D. S. Reeves, and P. Ning, "Defending against sybil attacks in sensor networks," 25^{th} *IEEE International Conference on Distributed Computing Systems Workshops*, 2005.