SCAN: Self-Organized Network-Layer Security in Mobile Ad Hoc Networks

Hao Yang, James Shu, Xiaoqiao Meng, Songwu Lu

***Abstract*— Protecting the network layer from malicious attacks is an important yet challenging security issue in mobile ad hoc networks. In this paper we describe SCAN, a unified network-layer security solution for such networks that protects both routing and data forwarding operations through the same reactive approach. SCAN does not apply any cryptographic primitives on the routing messages. Instead, it protects the network by detecting and reacting to the malicious nodes. In SCAN, local neighboring nodes collaboratively monitor each other and sustain each other, while no single node is superior to the others. SCAN also adopts a novel credit strategy to decrease its overhead as time evolves. In essence, SCAN exploits *localized collaboration* and *information cross-validation* to protect the network in a self-organized manner. Through both analysis and simulation results we demonstrate the effectiveness of SCAN even in a highly mobile and hostile environment.**

***Index Terms*— Self-organized security, mobile ad-hoc network, network-layer security.**

I. Introduction

An ad hoc network is a group of mobile wireless nodes that cooperate and forward packets for each other. Such networks extend the limited wireless transmission range of each node by multihop packet forwarding, thus well suited for the scenarios in which pre-deployed infrastructure support is not available, for example, emergency relief, military operations, and terrorism response. Security is one crucial requirement for these mission-critical applications.

In this paper we tackle an important security issue in ad hoc networks, namely the protection of their network-layer operations from malicious attacks. We focus on securing the packet delivery functionality because it is the premise for the multihop connectivity between two faraway nodes. Without appropriate protection, the malicious nodes can readily function as routers and prevent the network from correctly delivering the packets. For example, the malicious nodes can announce incorrect routing updates which are then propagated in the network, or drop all the packets passing through them. Several recent studies [1]–[4] have provided detailed description on such network-layer security threats and their consequences.

In ad hoc networks, multihop packet delivery is achieved through two closely related network-layer operations: ad hoc routing and packet forwarding. As a result, the security solution should encompass the protection of both. The secure ad hoc routing problem has been extensively researched and a number of secure routing protocols have been proposed in the literature, to name a few, Ariadne [4], SEAD [5], SRP [6], ARAN [7], and SAODV [8]. All these protocols focus on protecting the correctness of the routing table maintained at each node, while leaving packet forwarding largely unprotected. Moreover, they typically protect the routing messages through various cryptographic primitives, resulting in constant and non-trivial routing overhead in terms of both computation and communication. The companion key management problem is also challenging due to the self-organized nature of ad hoc networks [9]. On the other hand, the secure packet forwarding problem has received relatively little attention. While Watchdog and Pathrater [1] can mitigate the detrimental effects of packet drop in the context of DSR [10], its applicability in the distance-vector routing protocols, such as AODV [11] and SAODV [8], is not addressed yet. The fundamental problem is that, due to their strong interdependency, routing and packet forwarding should be protected together.

To this end, we present a network-layer security solution, called SCAN, that protects the control-plane (i.e., ad hoc routing) and the data-plane (i.e., packet forwarding) operations in a unified framework. SCAN does not apply any cryptographic primitives on the routing messages. Instead, it protects routing and packet forwarding through a same *reactive* approach, in which local neighboring nodes collaboratively sustain each other, monitor each other, and react to occasional attacks in their vicinity.

In SCAN, each node monitors the routing and packet forwarding behavior of its neighbors, and independently detects any malicious nodes in its own neighborhood. The monitoring mechanism takes advantage of the broadcast nature of wireless communication. In a network with reasonable node density, one node can often overhear the packets (including both routing updates and data packets) received as well as the packets sent by a neighboring node. In such cases, it can *cross-check* these packets to discover whether this neighbor behaves normally in advertising routing updates and forwarding data packets. We exemplify this idea in the context of AODV routing protocol [11], but its principal is applicable to other routing protocols as well (to be elaborated in Section VII). In order to enable such cross-checking, we modify the AODV protocol and add a new field, *next_hop*, in the routing messages, so that a node can correlate the overheard packets accordingly.

While each node monitors its neighbors independently, all nodes in a local neighborhood collaborate with each other to

Manuscript received October 1, 2004; revised August 15, 2005. This work was supported in part by NSF CAREER program under grant ANI-0093484 and in part by DARPA SensIT program under contract DABT63-99-1-0010.

H. Yang is with the Computer Science Department, University of California, Los Angeles, CA 90095 (email: hyang@cs.ucla.edu).

J. Shu is with the Northrop Grumman Corporation, San Pedro, CA, 90731 (email: James.shu@gmail.com).

X. Meng and S. Lu are with the Computer Science Department, University of California, Los Angeles, CA 90095 (email: xqmeng@cs.ucla.edu, slu@cs.ucla.edu).

eventually convict a suspicious node. This is achieved by a distributed consensus mechanism, in which a node is convicted only when its multiple neighbors have reached such a consensus. The motivation is that a single node may have inaccurate monitoring results due to node mobility, interference, channel error, etc., and the malicious nodes may intentionally accuse legitimate nodes. The distributed consensus mechanism significantly decreases the chance of falsely accusing a legitimate node, while maintaining a high probability of convicting the malicious nodes.

Once a malicious node is convicted by its neighbors, the network reacts by depriving its right to access the network. In SCAN, each node must possess a valid token in order to interact with other nodes and participate in the network. The token of a convicted malicious node will be revoked. We use asymmetric cryptography to prevent the forgery of tokens. Specifically, each token is signed by the same secret key so that it can be verified by a system-wide public key known to all nodes. We utilize a distributed mechanism similar to [3] in issuing and renewing the tokens. In this scheme, a group of nodes can collaboratively sign a token while no single node can do so, and each node renews the token from its neighbors once its current token expires.

To control the overhead of SCAN, we exploit a novel credit strategy in determining the token lifetime for each node. The more credits a node has, the longer time its token is valid for. A newly joined node has zero credit, hence is granted temporary admission into the network by obtaining a token that expires soon. The legitimate nodes are rewarded with credits at each time they successfully renew their tokens. Therefore, as time evolves, a well-behaving node renews its token less and less frequently by accumulating its credits. On the other hand, a malicious node is eventually detected by its neighbors and denied of network access as its token is revoked.

In essence, SCAN exploits two ideas to protect the mobile ad hoc networks: 1) *local collaboration*: the neighboring nodes collectively monitor each other and sustain each other; and 2) *information cross-validation*: each node monitors its neighbors by cross-checking the overheard transmissions, and the monitoring results from different nodes are further cross-validated. As a result, the security solution is self-organized, distributed, and fully localized.

We demonstrate the effectiveness of SCAN through both analysis and simulation results. We show that even if 30% of the nodes are malicious and the maximum mobility speed is $20m/s$, SCAN can detect 92% of the malicious nodes and increase the goodput by more than 150%. While maintaining this desired security strength in most scenarios, the overhead of SCAN gracefully adapts to the network status, such as node mobility speed and the number of malicious nodes. However, one drawback of SCAN is that the legitimate nodes also have a non-zero, though quite small, probability of being incorrectly accused. Both analysis and simulation results show that there is a fundamental tradeoff between the detection power and the false accusation probability.

The rest of this paper is organized as follows. Section II provides the background on the AODV routing protocol. Section III formulates the network and security models. Section IV describes the SCAN design in details. Section V analyzes the overhead of SCAN, and Section VI presents the simulation evaluation using *ns-2* network simulator. Section VII explains our design rationale and discusses several important issues. Section VIII reviews and compares to the related work, and Section IX concludes the paper.

II. AODV Routing Protocol

We briefly overview the AODV routing protocol [11] in this section. AODV has been one of the most popular on-demand ad hoc routing protocols studied in the research community and IETF [12]–[14]. In Section IV-B, we will use it as the context to illustrate how to monitor routing behavior.

The path discovery process in AODV is entirely on-demand. When a source node needs to send packets to a destination to which it has no available route, it broadcasts a RREQ (Route Request) packet to its neighbors. Each node maintains a monotonically increasing sequence number to ensure loop-free routing and supersede stale route cache. The source node includes the known sequence number of the destination in the RREQ packet. The intermediate node receiving a RREQ packet checks its routing table entries. If it possesses a route toward the destination with greater sequence number than that in the RREQ packet, it unicasts a RREP (Route Reply) packet back to its neighbor from which it received the RREQ packet. Otherwise, it sets up the reverse path and then rebroadcasts the RREQ packet. Duplicate RREQ packets received by one node are silently dropped. This way, the RREQ packet is flooded in a controlled manner in the network, and it eventually arrives at the destination itself or a node that can supply a fresh route to the destination, which then generates the RREP packet. As the RREP packet is propagated along the reverse path to the source, the intermediate nodes update their routing tables using distributed Bellman-Ford algorithm with additional constraint on the sequence number, and set up the forward path.

AODV includes a path maintenance mechanism to handle the dynamics in the network topology. Link failures can be detected by either periodic beacons or link layer acknowledgments, such as those provided by 802.11 MAC protocol [15]. Once a link is broken, an unsolicited RREP packet with a fresh sequence number and infinite hop count is propagated to all active source nodes that are currently using this link. When a source node receives the notification of a broken link, it may restart the path discovery process if it still needs a route to the destination.

III. Models and Assumptions

In this section we formulate the network model and the security model, then describe our design assumptions.

A. Network Model

We consider a wireless mobile ad hoc network consisting of an unconstrained number of networking nodes. Each node may freely roam, or remain stationary in a location for an extended period of time. In addition, each node may join the network, leave the network, or fail at any time. The nodes

perform peer-to-peer communication over shared, bandwidth-constrained, error-prone, and multi-hop wireless channel. For differentiation purpose, we require each node to have a unique non-zero ID.

The communication in the network is bi-directional, i.e., two nodes within the wireless transmission range may communicate with each other. This is also required by most wireless Medium Access Control (MAC) protocols such as 802.11 [15]. However, the wireless channel may be lossy, asymmetric, and prone to interference, as shown in the recent measurements results [16].

We assume that each node's wireless interface may operate in the promiscuous mode in the link layer, that is, it can overhear ongoing communications within its wireless transmission range. Most existing 802.11-based wireless cards can readily support such a promiscuous mode. In practice, security and privacy concerns may arise when a node can overhear packets that are not destined to itself. This is typically addressed by end-to-end or link-layer encryption. We stress that our design does not require the nodes to understand the semantics of the overheard data packets, thus can work well in the presence of various encryption mechanisms. The promiscuous mode may also incur extra computation overhead and energy consumption in order to process the transit packets. However, the energy efficiency issue is out of the scope in this work.

B. Security Model

In an unprotected mobile ad hoc network, a malicious node may readily participate as a router and disrupt the network-layer packet delivery functionality. Because multihop packet delivery is achieved through ad hoc routing on the control plane and packet forwarding on the data plane, attacks on either of them can lead the network into malfunction. In this section, we explicitly distinguish the vulnerabilities in routing and packet forwarding, and consider a generic network-layer attack model. In this model, the malicious nodes can launch any or both of two broad categories of attacks: *routing misbehavior* and *packet forwarding misbehavior*.

The *routing misbehavior* refers to any action of advertising routing information that does not follow the specifications of the routing protocol. The maneuver that the malicious nodes may take is protocol-dependent. In the context of AODV, a malicious node may advertise a route with a distance metric smaller than its actual distance to the destination; it may advertise a route with a large sequence number and invalidate all routing updates from other nodes; it may also initiate routing discovery very frequently to waste the network resource [8]. Multiple colluding attackers may create route loops by introducing shortcuts in the network, known as *wormhole* attacks [17]. Consequently, the malicious nodes can force a source node to use a "dangerous" route which is under their control. The legitimate nodes are prevented from discovering optimal routes or in the worse case, any available route.

On the other hand, the *packet forwarding misbehavior* refers to any intentional disruption of the data forwarding activity. This is independent of the underlying routing protocol. For example, the malicious nodes along an active route may drop the packets, resend the (altered) packets, or inject other packets. They may also pump lots of dummy packets into the network as a brute-force form of network-layer denial-of-service (DoS) attack. Furthermore, they may adopt more tricky strategies, for example, dropping the packets in a probabilistic manner instead of blindly dropping all packets. As a result, the packets from legitimate nodes cannot reach the destination even if a route has been correctly established. The network resource is wasted, and severe network congestion and channel contention may occur in the network.

In this work we consider an attacker who can perform any combination of attacks that are within the above generic attack model. We do not address passive attackers who eavesdrop and record the wireless transmission. We assume that multiple attackers may co-exist in the network, and several attackers may even collude with each other. However, we assume that each group of colluding attackers has less than k nodes, where k is a design parameter (to be introduced in IV-A). We also assume that the attacker cannot impersonate a legitimate node by forging its ID, which can achieved through existing message authentication mechanisms [15], [18].

We do not address node selfishness in this work. We refer to recent publications [19]–[21] on how to stimulate cooperation in an ad hoc network. We largely neglect security threats in the physical layer and the link layer. Such lower-layer attacks can be limited by mechanisms such as the spread-spectrum technology, the WEP protocol, and MAC misbehavior handling mechanisms [22].

IV. SCAN Design

In this section we present the SCAN design in details. In order to protect the packet delivery functionality, each SCAN node overhears the wireless channel in the promiscuous mode, and monitors the routing and packet forwarding behavior of its neighbors at all time. The monitoring results at different nodes in a local neighborhood are cross-validated. A malicious node is convicted when its neighbors have reached such a consensus, then it is deprived of the network membership and isolated in the network. In order to enforce the network access, each legitimate node carries a valid token which certified, unexpired, and not revoked, while any node without a valid token is denied of participation in the network operations. A legitimate node can always renew the token from its neighbors before its current token expires. However, when a malicious node is convicted, its neighbors collectively revoke its current token and inform all other nodes in the network. The above SCAN framework is illustrated in Figure 1, which has the following three components:

- *Collaborative Monitoring*: all nodes within a local neighborhood collaboratively monitor each other.
- *Token Renewal*: all legitimate nodes in a local neighborhood collaboratively renew the tokens for each other.
- *Token Revocation*: the neighbors of a malicious node, upon consensus, collaboratively revoke its current token.

In this framework, the malicious nodes are detected and convicted via the collaborative monitoring mechanism, which collects and analyzes each node's behavior in the routing and

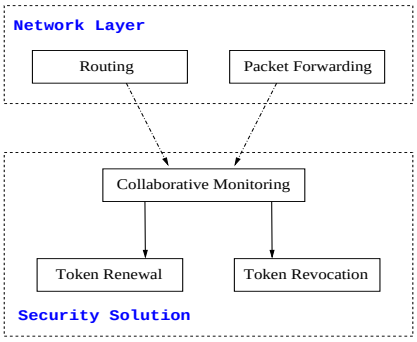

Fig. 1. The SCAN framework: components and interactions.

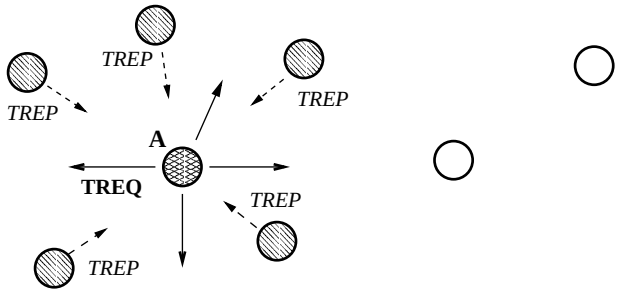

Fig. 2. Message handshake in the localized token issuing process.

packet forwarding activities. The token revocation mechanism reacts to occasional attacks launched by the malicious nodes by revoking their tokens and alerting the network. This proactively prevents a convicted attacker from further disrupting the network operations, because without a valid token, it can not participate in the network any more. The token renewal mechanism ensures that legitimate nodes can continue to stay in the network by renewing their token from time to time.

The above seemingly simple operations pose several research challenges. For example, how can we provide "anytime, anywhere" token renewal with low overhead? How can each node monitor the behavior of its neighbors? How accurate the monitoring results are, and how can we improve this accuracy? Given that the malicious nodes may roam in the network, what is the effective way to isolate them in the network? The remaining of this section will address each of these questions in details. We start with the token renewal process in Section IV-A, and present the collaborative monitoring mechanism in Section IV-B. Token revocation is described in Section IV-C. Finally we summarize the SCAN design in Section IV-D.

A. Token Renewal

SCAN implements its token renewal operations based on an earlier proposal of distributed certification service for mobile ad hoc networks [3]. In order to communicate with other nodes in the network, each legitimate node carries a token which contains the following three fields <*owner_id*, *signing_time*, *expiration_time*>. The tokens are protected by the public-key cryptographic primitives. There is a single key pair PK/SK in the network. The public key PK is known to all nodes when they join the network, while the secret key SK is used to sign each token. Since the token is certified and bound to the owner's unique ID, a malicious node can not fabricate a token or steal the token from another legitimate node.

In [3], no single node knows the secret key SK, hence has the authority of signing the tokens. Instead, such an authority is distributed equally into each node in the network. This is realized by exploiting a polynomial secret sharing scheme[1], in which each node shares the secret key SK by a polynomial of order $k-1$, where k is a design parameter. As a result, a group of k nodes can collaborate to sign a token with SK, but a group of less than k nodes can never do so. This way, the certification service can resist up to $k-1$ colluding malicious nodes in the network.

Before the current token expires, each node solicits its local (typically one-hop or two-hop) neighbors to renew its token. The message handshake in this localized token renewal process is illustrated in Figure 2. The node that needs token renewal broadcasts a TREQ (Token Request) packet, which contains its current token and a timestamp. As we will describe in Section IV-C, each node keeps a Token Revocation List (TRL) based on the token revocation mechanism. When a node receives a TREQ packet, the TRL is used to decide whether to serve the request or not.

Specifically, when a node receives a TREQ packet from its neighbor, it extracts the token from the packet. It checks whether the token has already been revoked by comparing it with the TRL. If the token is still valid yet about to expire, it constructs a new token with *owner_id* equal to that in the old token, *signing_time* equal to the timestamp in the TREQ packet. The *expiration_time* is determined by the credit strategy described below. It then signs the newly constructed token using its own share of SK, encapsulates the partially signed token in a TREP (Token Reply) packet, and then unicasts the TREP packet back to the node from which it received the TREQ packet. TREQ packets containing revoked tokens are silently dropped. When the requesting node receives k TREP packets from different neighbors, it combines these partially signed tokens into a single token signed with SK.

1) Credit Strategy in Token Lifetime: Now we consider how to determine the token lifetime, i.e., the *expiration_time* field in a token. Since the token must be renewed once it expires, the legitimate nodes may be penalized by the computation and communication overhead associated with the token renewal process. From this perspective, one might think that long lifetime for tokens be desirable. However, once a token with long lifetime is revoked, it has to be kept by each node in its TRL for a long period of time until it expires, resulting in an increased length of the TRL. Therefore, the token lifetime represents a tradeoff between the overhead and the number of states kept at each node.

We propose a novel credit strategy to determine the token lifetime, which can decrease the token renewal overhead as time evolves yet keep the TRL length bounded by a constant factor. In this strategy, a newly joined node is issued a token with short lifetime. It accumulates its credit when it remains to

[1]Due to space limits, we refer to [3] for the cryptographic implementation of this scheme.

behave well in the network, and its subsequent token lifetime depends on its credit at the renewal time. The more credit one node has, the longer lifetime its token has. This way, a legitimate node will have its token lifetime steadily increased over time, thus renewing its token less and less frequently.

Our credit strategy is implemented by additively increasing the token lifetime each time a node renews its token. Let T_1, T_2, T_3, T_4 denote the *signing_time* and *expiration_time* fields in the previous and renewed tokens, respectively. The additive increase algorithm simply states that $T_4 - T_3 = T_2 - T_1 + T_0$. That is, each time a legitimate node renews its token, its token lifetime increases by T_0.

With the assumption that the probability of a node being an attacker is reciprocal to the duration of the time it has stayed in the network with well behavior[2], we can show the benefit of the credit strategy by comparing it to the constant lifetime strategy, which always sets the token lifetime as T_0.

In the credit strategy case, when a node receives its n-th token, the duration of the time it has stayed in the network[3] is $T_L = \sum_{i=1}^{n-1} iT_0 = \frac{n(n-1)T_0}{2}$. Thus, a node that has stayed in the network for a duration of time T needs to renew its token for $N_1 \approx \sqrt{\frac{2T}{T_0}}$ times. On the contrary, in the constant lifetime strategy case, the same node has to renew its token for $N_2 = \frac{T}{T_0}$ times. We can see that $N_2 \approx \frac{N_1^2}{2}$, which demonstrates the significant savings in the token renewal overhead.

While the credit strategy gradually increases the token lifetime for the long-lived and well-behaving nodes, it does not impose heavy burden on the revocation states that each node has to keep. In fact, the expected time of a node's n-th token kept in the TRL is

$$\begin{aligned} T_C &= \int_{T_L}^{T_L+nT_0} \frac{T_L + nT_0 - t}{t} \mathrm{d}t \\ &< \int_{T_L}^{T_L+nT_0} \frac{T_L + nT_0 - t}{T_L} \mathrm{d}t = \frac{nT_0}{n-1} \end{aligned} \quad (1)$$

which is asymptotically bounded by T_0. Hence, the expected length of the TRL is also bounded by a constant number. In essence, the credit strategy takes advantages of the characteristics of node behavior, and rewards well-behaving nodes by decreasing their token renewal overhead.

2) Avoid Synchronization of Token Renewal: In order to avoid synchronized token renewal requests among the nodes, we introduce randomization in the timers that they associate with such requests. Let T_s and T_e denote the *signing_time* and *expiration_time* fields in a node's current token, respectively. Instead of requesting token renewal exactly before T_e, the node randomly picks up a value $\hat{T}_e$ with uniform distribution over $[0.2 * T_s + 0.8 * T_e, T_e]$, and broadcasts the TREQ packet at time $\hat{T}_e$.

[2]This assumption is motivated by the analogy of how the credit card companies set up the credit line for their customers. We admit that it is a simplified model for the user behavior. However, it indeed reflects some characteristics of the attackers in that they usually will not stay and behave well in the network for a long time.

[3]For simplicity of representation, we assume that the period of validity of the first token is T_0.

B. Collaborative Monitoring

The collaborative monitoring mechanism in SCAN monitors the routing and packet forwarding operations of each node in a fully decentralized and localized manner. Each node overhears the channel, monitors the behavior of its neighbors, and discovers consistent misbehavior as indications of attacks. Moreover, local neighboring nodes collaborate with each other to improve the monitoring accuracy. We exemplify this mechanism in the context of AODV routing protocol [11]. However, as we shall discuss in Section VII, it can be easily extended to accommodate other ad hoc routing protocols. Below we first describe how a single node monitors its neighbor's routing and packet forwarding behavior in Section IV-B.1 and Section IV-B.2 respectively, then introduce distributed collaborative consensus in Section IV-B.3.

1) Monitor Routing Behavior: Our basic idea is to overhear the channel and *cross-check* the routing messages announced by different nodes. This can be applied to any distributed and deterministic routing protocol. In such protocols, the routing activity of a node is a three-step process : a) receiving routing updates from neighboring nodes as inputs to a routing algorithm; b) executing the routing algorithm; c) announcing the output of the routing algorithm as its own routing updates. The monitoring task is to verify whether the routing algorithm executed by a node follows the protocol specifications. In other words, the trustworthiness of a routing message, as the output of the routing algorithm, can be examined when the monitoring node knows the input to the algorithm, since the algorithm itself is publicly known and deterministic.

We exemplify this idea in the context of AODV, in which the routing algorithm is essentially the distributed Bellman-Ford algorithm with constraints on sequence number. Unfortunately, by overhearing a routing update, an AODV node cannot obtain enough information about the routing algorithm's input on the advertiser side. The key reason is that the *next hop* information is missing in the AODV routing messages. Thus, when a node announces a routing update, its neighbors have no clue about which node is the next hop in the route, and hence cannot judge on its input to the routing algorithm, i.e., the original routing update on which its routing computation is based.

In order to enable the cross-checking of routing updates, we make two modifications to AODV. First, we add one more field, *next_hop*, in the RREP packet. Similarly, we add one more field, *previous_hop*, in the RREQ packet. This way, each node explicitly claims its next hop in a route when it advertises routing updates. Secondly, each node keeps track of the routing updates previously announced by its neighbors. Essentially each node maintains part of the routing tables of its neighbors. This redundancy of routing information makes it possible for a node to examine the trustworthiness of future routing updates from its neighbors.

Figure 3 illustrates how a node can cross-check the routing updates and examine their trustworthiness. Suppose that node M is a neighbor of, and hence monitors, both node X and node Y. M has kept track of the route entries previously announced by Y. When M receives a new routing update from X which claims Y as the next hop, M can examine this update by

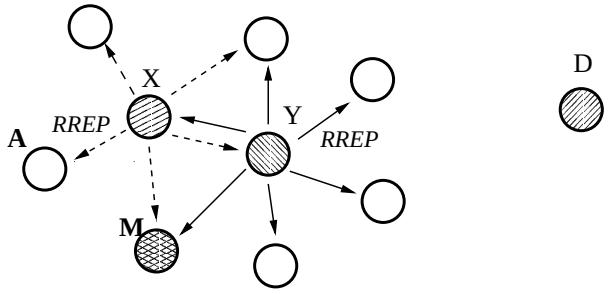

Fig. 3. Each node can monitor the routing behavior of its neighbors by cross-checking the overheard routing updates.

comparing it with the route entry announced by Y earlier. We can view this checking process as if M were reconstructing the routing algorithm execution performed by X. Because the two executions (one is the real execution performed by X, another is the "virtual" execution performed by M) have the same input into the same algorithm, they should generate the same results. Any inconsistency between them means that X did not correctly follow the protocol specifications.

Take one field, *hop_cnt*, as an example. Suppose that Y previously announced a route toward destination D with *hop_cnt* as 2. Now X announces a new routing update toward D with *hop_cnt* as 1, claiming that its next hop is Y. M can readily detect this routing misbehavior, because based on the route announced by Y, it can predict the correct distance from X to D via Y to be 3. The same idea can be applied to examine other fields in the routing updates as well.

The cross-checking of routing updates is only performed at the common neighbors of one node and its next hop node. Consider again the scenario in Figure 3. When A receives the routing update from X, it cannot tell whether this update is trustworthy or not, because it has no information about the next hop node, Y, in the offered route. In this case, A skips the cross-checking process.

The proposed routing misbehavior monitoring mechanism avoids cryptographic operations on the routing messages. Compared to the secure routing approach, this crypto-free feature can significantly reduce the computation and communication overhead. However, it also has disadvantages in that it might not work well in several cases: 1) Y has only stayed in M's neighborhood for a short period of time due to mobility, and M has not recorded all the route entries announced by Y; 2) M did not receive the previous route updates sent by Y due to channel error; 3) Y has increased the lifetime of a route entry, but M is unaware of this change and has deleted it from its cache. We rely on the collaborative consensus mechanism (Section IV-B.3) to enhance the monitoring performance.

2) Monitor Packet Forwarding Behavior: Each SCAN node also monitors the packet forwarding activity of its neighbors. This is achieved by overhearing the channel and comparing ongoing data transmission with previously recorded routing messages. We currently focus on three kinds of forwarding misbehavior, namely packet drop, packet duplication, and network-layer packet jamming, and develop simple algorithms to detect each of them. Packet drop means that a node drops the packets that it is supposed to forward for its neighbors; packet duplication means that a node duplicates the packets that it has already forwarded; and packet jamming means that a node sends too many packets and occupies a significant portion of the channel bandwidth.

The packet drop detection algorithm is similar to the *watchdog* technique in [1]. However, *watchdog* was originally proposed to work with DSR [10], in which the sender explicitly lists the route in the data packet header. It cannot be directly applied in the AODV context, because when a node receives a packet, its neighbors do not know to which node it should forward the packet, thus cannot tell whether it forwards the packet in the correct manner. Fortunately, our modification to the AODV protocol, described in the previous section, enables the detection of packet drop, because each node keeps track of the route entries announced by its neighbors, which explicitly lists the *next_hop* field.

Specifically, each SCAN node records the headers of the recent packets it has overheard. When it overhears one packet sent to a neighbor, say X, it checks the cache of the route entries announced by X, and determines the next hop node to which X should forward the packet. If it has not overheard the packet being forwarded by X to the correct next hop node after certain time, it considers this packet as being dropped. If the bandwidth corresponding to the packets dropped by X exceeds a threshold $Drop\_Bandwidth$, it considers this as misbehavior in the packet forwarding service.

The detection of packet duplication or jamming follows the similar structure. If one node overhears that the bandwidth consumed by duplicate packets from its neighbor exceeds the threshold $Duplicate\_Bandwidth$, or the bandwidth consumed by packets originated from its neighbor exceeds the threshold $Sending\_Bandwidth$, it also considers these events as packet forwarding misbehavior.

The localized monitoring mechanism performed by individual node is intrinsically inaccurate due to the inaccuracy in the information obtained from channel overhearing. The detection accuracy is sensitive to multiple factors, such as channel error, mobility, parameters in the detection algorithm, etc. It is also susceptible to *blackmail* attacks, in which an attacker blackmails its legitimate neighbors as misbehaving nodes. Next we describe a distributed collaborative consensus mechanism that exploits the collaboration among local neighboring nodes to improve the monitoring performance.

3) Distributed Collaborative Consensus: In the collaborative consensus mechanism, local neighboring nodes collaborate with each other to *cross-validate* the monitoring results at different nodes and reach a consensus. We use "m out of N" strategy as the consensus criteria. That is, a node is considered as an attacker if and only if m nodes out of all its N neighbors have independently detected its misbehavior.

The "m out of N" strategy can significant improve the monitoring performance, which can be quantitatively evaluated by two metrics: detection probability (correct detection of an attacker) and false alarm probability (false accusation against a legitimate node). Let P_1 and P_2 denote the detection and false alarm probability of individual monitoring results, respectively.

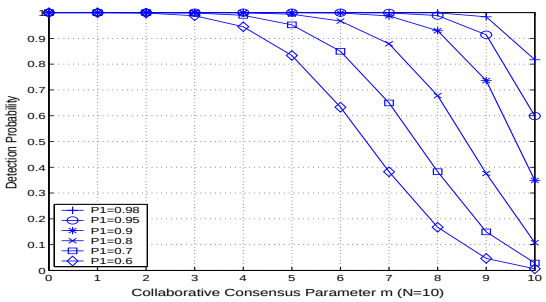

Fig. 4. Increasing detection probability by collaborative consensus.

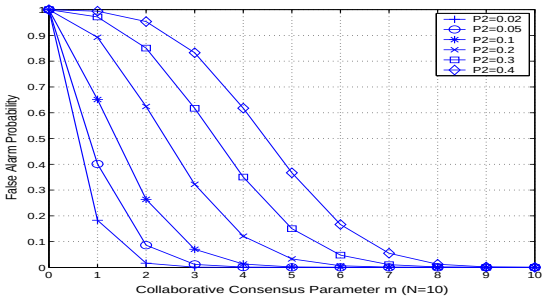

Fig. 5. Decreasing false alarm probability by collaborative consensus.

With collaborative consensus, the detection probability is:

$$P_D = \sum_{k=m}^{N} \binom{N}{k} P_1^k \ (1-P_1)^{N-k} \tag{2}$$

Meanwhile, the false alarm probability is:

$$P_F = \sum_{k=m}^{N} \binom{N}{k} (1-P_2)^k \ P_2^{N-k} \tag{3}$$

The above two equations are visualized in Figures 4 and 5, where we fix N as 10 and vary the choice of m. We can see that by choosing an appropriate value for m, one can increase detection probability P_D and decrease false alarm probability P_F simultaneously. There are several approaches to selecting m as a function of N, such as $N/2$, a constant number k (the secret sharing parameter), or a value that guarantees both P_D and P_F are within certain range. The selection of m represents a tradeoff between the prompt reaction to the attackers and the protection of legitimate nodes from false accusation. We will study the impact of different schemes in future work.

The collaborative consensus mechanism is implemented in a distributed manner. Each node broadcasts a SID (Single Intrusion Detection) packet once it detects the misbehavior of any neighbor. We do not differentiate the SID packets triggered by routing and packet forwarding misbehavior. When a node has received m independent SID packets against the same node, it constructs a notification of token revocation, signs the notification using its own share of SK, encapsulates the signed notification in a GID(Group Intrusion Detection) packet, and then broadcasts the GID packet. When a node has received k GID packets, it constructs a TREV (Token Revocation) packet signed by the SK, using the same polynomial secret sharing primitive as we described in the token renewal process.

C. Token Revocation

Now we describe how SCAN revokes a malicious node's token in the network. Recall that each SCAN node keeps a TRL (Token Revocation List). The token revocation process is initiated when a constructed TREV packet is broadcasted. When a node receives a TREV packet, it checks whether the packet is signed by SK, and whether the revoked token is already on the TRL. TREV packets that are not signed by SK or contain tokens on the TRL are silently dropped. Otherwise, it adds the revoked token into its own TRL and rebroadcasts the TREV packet. This way, eventually every node will add the revoked token into its TRL. Moreover, any active link from the revoked token's holder is deemed as broken and canceled out by the path maintenance mechanism in the routing protocol.

Because only nodes with valid tokens can participate in the network operations, the token revocation mechanism ensures that a malicious node is isolated right after it was detected. While the TREV packet is essentially flooded in the network, the associated communication overhead is affordable because there is only one TREV packet per attacker.

Each TRL entry is also associated with a *soft-state* timer. In order to ensure that a malicious node cannot renew its token, a revoked token has to be kept in TRL until it expires, after which it can be deleted. This soft state reduces both the storage overhead and the processing overhead when a node checks the validity of the tokens presented by its neighbors.

D. Summary

So far we have described the SCAN design in detail. SCAN is self-organized in that all nodes in the network equally participate in the security solution: each node shares a portion of a global secret, monitors the behavior of its neighbors, and renews the token for its neighbors. It does not assign any special role to a single node, or assume that the nodes are equipped with *a priori* trust relationship or secret association. Instead, it exploits secret sharing techniques to enhance the tolerance to compromised nodes and withstand limited collusion among the attackers. As a result, it always trusts a group of nodes collectively without completely trusting any individual node. The SCAN design is also fully localized as all its basic operations are performed in the local neighborhood. In essence, SCAN exploits extensive collaboration among local neighboring nodes in protecting the network layer.

V. Overhead Analysis

In this section, we analyze the *storage*, *computation*, and *communication* overhead of SCAN, and provide a simple yet meaningful overhead comparison between the reactive and the proactive approaches.

A. Model and Notation

We consider a mobile ad hoc network in which N nodes are uniformly distributed in the field. The average number of neighbors within a node's wireless communication range is D. The communication overhead to flood an area is proportional to the number of nodes in it, with a constant ratio of α. There

are q nodes in the network that are malicious, denoted by $M_1, M_2, \cdots, M_q$. For simplicity, we do not consider node arrival and departure in this analysis.

The network lifetime, i.e., the duration of time that the network operates, is T. Each node is initially assigned a token with lifetime $T_0 << T$. When its current token expires, a node renews the token with lifetime increased by T_0 (see Section IV-A.1). A malicious node M_i starts to launch the attacks at time S_i, and its neighbors reach a consensus to revoke its token at time $E_i > S_i$. The utilization of the network is measured by the average number of route requests sent out during one time unit, denoted by r.

B. Storage Overhead

The storage overhead of SCAN comes from two sources. First, for token renewal and revocation purposes, each SCAN node keeps the tokens of all its D legitimate neighbors, and a TRL that maintains the current revoked yet unexpired (at most q) tokens. In order to perform the secret sharing cryptographic primitives, each SCAN node also keeps the system public key PK, and its own share of the system secret key SK.

Secondly, for monitoring purpose, each SCAN node keeps the overheard routing entries advertised by its neighbors. The number of entries is DN in the worst case. However, in the context of on-demand routing protocols such as AODV, it is quite reasonable to expect this number to be much smaller than DN, say $O(N)$. In addition, each node records the recent data packets that it has overheard. This cache has a constant size, bounded by the production of the bandwidth and the time that an old packet is kept. It can be further reduced by compressing the packets using a hash function.

Therefore, the overall storage overhead of SCAN is $O(DN)$ in the worst case, and $O(N)$ in a light- or medium-loaded network with on-demand routing protocols.

C. Computation Overhead

Due to the vast variety of cryptographic algorithms and their implementation, we measure the computation overhead using a generic metric: the number of cryptographic primitive executions, while neglecting the details (e.g., number of CPU instructions) of each primitive. We also neglect the computational overhead of the monitoring mechanism, which requires only table lookup and simple comparison.

The only cryptographic primitives in SCAN are the polynomial secret sharing used in token manipulation, namely token renewal and revocation. Each renewed token involves $D+1$ cryptographic computations at most (one computation at the requesting node and one at each of its D neighbors). Similarly, each revoked token involves $D+1$ cryptographic computations at most. As we show in Section IV-A, each legitimate node renews its token for $\sqrt{2T/T_0}$ times. Each malicious node is revoked of its token only once. Therefore, the total number of cryptographic primitive executions in the entire network, throughout the network lifetime, is:

$$CPO_{SCAN} = (q + N\sqrt{\frac{2T}{T_0}})(D+1) \tag{4}$$

In contrast, the proactive approach (e.g., [4]–[7]) seeks to prevent the malicious attacks by applying cryptographic primitives (e.g., digital signatures, or Message Authentication Codes) on the routing messages. As a result, each time a node receives a routing update, it has to perform two cryptographic computation: one to verify the received update, the other to generate its own update. As a conservative estimation, we consider only the computation overhead associated with processing route request packets, each of which is flooded in the network. Thus, the total number of cryptographic primitive executions in the proactive approach is at least:

$$CPO_{Proactive} = 2rT\alpha N \tag{5}$$

To compare SCAN with the proactive approach, we have:

$$\frac{CPO_{SCAN}}{CPO_{Proactive}} \approx \frac{D+1}{\alpha r\sqrt{2TT_0}} = O(\frac{1}{\sqrt{T}}) \tag{6}$$

We can see that in an ad hoc network with long operational lifetime, SCAN has *asymptotically lower* computation overhead compared with the proactive approach that authenticates each routing message. This is because SCAN performs computation-intensive cryptographic computation only on token manipulation, which happens much less frequently than routing message exchange.

For example, consider an ad hoc network that has $N = 100$ nodes and operates for 2 hours (i.e., $T = 120$ minutes). Each node initially has a token with lifetime $T_0 = 10$ minutes. On average, each node has $D = 10$ neighbors in its transmission range, and initiates one data transmission every 10 minutes. Thus, $r = N/10 = 10$ (routing requests per minute). We further assume $\alpha = 1$ for simplicity. Based on equation 6, we can see that $\frac{CPO_{SCAN}}{CPO_{Proactive}} \approx 0.02$, which shows that the computation overhead of SCAN is significantly lower than the proactive approach in this network setup.

We note that this analysis uses the total number of cryptographic primitive executions in measuring the computation overhead. This simple metric may deviate from the actual measurement in terms of CPU cycles, etc. Different primitives may also have significantly varying computational characteristics, which is determined by both the cryptographic algorithm and its implementation. For example, a hash function typically requires much less CPU cycles to compute than an asymmetric cryptographic primitive such as RSA encryption. We leave an extensive study on this aspect to future work.

D. Communication Overhead

The communication overhead of SCAN mainly comes from token renewal, collaborative monitoring, and token revocation. Similar to the previous analysis of computation overhead, each renewed token involves one TREQ packet and D TREP packets at most. The communication overhead of collaborative monitoring depends on the detection time $E_i - S_i$. During this time period, the neighbors of the malicious node M_i detect its misbehavior individually, broadcast these detection results in SID packets, then reach the consensus. If the detection time is too long, the SID packets may be re-broadcasted. Otherwise, for a malicious attacker, at most D SID packets

and D GID packets are locally broadcasted. Finally, the TREV packet is constructed and then flooded in the network. Suppose the average size of the TREQ, TREP, SID, GID, and TREV packets is C_1, then the communication overhead of SCAN is:

$$CMO_{SCAN} = (D+1)N\sqrt{\frac{2T}{T_0}}C_1 + q(2D+\alpha N)C_1 \quad (7)$$

In the proactive approach, instead, the communication overhead comes from the increased length of the routing messages, which are appended with digital signature or Message Authentication Code. The length of the appended digest differs from one solution to another, and it may further change as the routing message traverses different nodes in the network. We neglect the details of these solutions, and denote the average length of the appended digest as C_2. Similar to the previous computation overhead analysis, the communication overhead of the proactive approach is at least:

$$CMO_{Proactive} = rT\alpha NC_2 \quad (8)$$

The comparison between SCAN and the proactive approach follows:

$$\frac{CMO_{SCAN}}{CMO_{Proactive}} \approx \frac{(D\sqrt{2T/T_0}+\alpha q)C_1}{\alpha rTC_2} = O(\frac{1}{\sqrt{T}}) \quad (9)$$

which shows that SCAN has *asymptotically lower* communication overhead compared with the proactive approach in a long-lived ad hoc network. This benefit originates from the fact that SCAN avoids increasing the length of frequently transmitted routing messages, and amortizes the communication overhead of collaborative detection over a large time window of network lifetime.

In summary, SAN has consistently lower computation and communication overhead than the proactive approach when the network continues to operate for a long period of time. Nevertheless, we point out that this is achieved at the cost of temporary disruption of network operations during the detection phase. An extensive performance evaluation of SCAN should take into account the detection speed and accuracy as well; however, this is out of our focus in this analysis.

VI. Simulation Evaluation

In this section we evaluate the performance of SCAN through extensive simulations, the goal of which is to answer the following questions:

- How well can SCAN detect and isolate a malicious node in the network?
- How well can SCAN protect a legitimate node from being incorrectly accused?
- How well can SCAN protect the network-layer packet delivery functionality?
- How large is the overhead introduced by SCAN?
- Which factors may affect SCAN's performance, and how?

We start with the simulation methodology and performance metrics in Section VI-A, then evaluate the performance of SCAN from the above aspects in Section VI-B to VI-D. The results show that SCAN is effective in protecting the network layer of ad hoc networks even in a highly mobile and hostile environment.

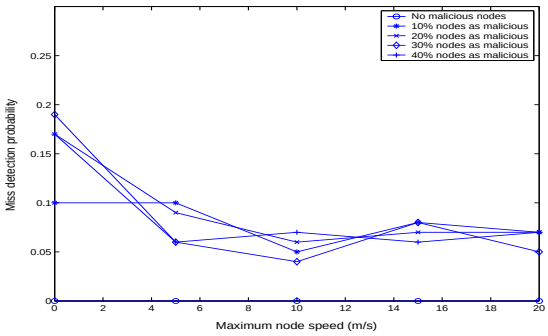

Fig. 6. Miss detection probability vs. Mobility

A. Methodology and Metrics

We have implemented SCAN in the *ns-2* simulator. Our performance evaluations are based on the simulations of 200 wireless nodes that form an ad hoc network over a rectangular (3000m $\times$ 600m) flat space in 1500 seconds of simulation time. The physical layer at each networking interface is chosen to approximate the Lucent WaveLAN wireless card. The MAC layer protocol and the routing protocol are 802.11 DCF and modified AODV protocol (Section IV-B), respectively. We use an improved version of "random waypoint" model [12], which is recently proposed in [23], as the mobility model. We set the minimum speed for each node as 2 m/s except for the static network case, and vary the maximum speed to evaluate the impact of node mobility on SCAN performance. The pause time is set to 0 to simulate an ad hoc network in which nodes are constantly roaming.

Before the simulation runs, we randomly select a certain fraction, ranging from 0 to 40%, of the network population as malicious nodes. Each malicious node picks up a random subset from the pool of possible attacks as its action strategy in the simulations. The attack pool includes all misbehavior that we have described in Section IV, for example, modifying the *hop_cnt* or *seq_number* fields in the routing updates (routing misbehavior), dropping or duplicating the data packets, blasting lots of packets (packet forwarding misbehavior). It is possible that a malicious node may select a combination of different misbehavior strategies. In the simulation run, multiple random UDP CBR traffic is sent in the network, each starting at a random time and lasting until the simulation terminates. We have varied the number of CBR connections from 10 to 30 and the simulation results all follow the same trend. For simplicity, we present only the results where 10 CBR traffic is sent. The legitimate nodes participate in the routing and packet forwarding activities in a normal manner, i.e., following all protocol specifications. On the contrary, the malicious nodes attempt to disrupt the network operations according to their pre-selected strategy.

In the simulations we are interested in the following metrics: 1) *miss detection ratio*, which is the chance that SCAN fails to convict and isolate a malicious node; 2) *false accusation ratio*, which is the chance that SCAN incorrectly convicts and isolates a legitimate node; 3) *packet delivery ratio*, which is the percentage of packets that are successfully delivered to the

receiver nodes; 4) *communication overhead*, which is the total number of packets sent by SCAN in order to achieve its goal.

Note that in a specific simulation run, due to the constraints of the dynamic network topology, some malicious nodes may not have the chance to realize their pre-selected attack strategy. For example, a malicious node that plans to drop the data packets can only do so when it resides in an active route. We define "active" malicious nodes as those that have indeed misbehaved in the network operations, no matter how short the mis-behaving time period is. For fairness purpose, we obtain the miss detection ratio by considering only the set of active malicious nodes, instead of all pre-chosen malicious nodes. The false accusation ratio is obtained in a similar way over the set of active legitimate nodes.

B. Monitoring and Detection

Now we evaluate the detection performance of the collaborative monitoring mechanism in SCAN in terms of miss detection and false accusation ratios. Recall that the collaborative consensus mechanism adopts a "m out of N" strategy (Section IV-B.3), in which m is an important parameter that can tradeoff between the miss detection ratio and the false accusation ratio. In these simulations we fix m as 6 because on average two neighboring nodes have about 10 common neighbors, and study the impact of mobility and the number of malicious nodes.

Figure 6 shows the miss detection ratio as the node mobility speed changes. We can see that this ratio is the highest in a static network, regardless of the number of malicious nodes. The miss detection ratio drops considerably when nodes start to move, and remains stable at $4-8\%$ when the speed further increases. We discover from the simulation traces that SCAN fails to convict a malicious node mainly because it resides in a sparsely occupied region. In such cases, there are not enough legitimate nodes in its neighborhood to reach a consensus. This also explains why mobility helps to improve the detection performance. In a static network, if a malicious node happens to stay in a sparsely occupied region, its neighbors always have no chance to convict it. On the contrary, in a mobile network, the mobility increases the chance that other nodes roam into this region or the malicious node itself moves into another densely occupied region. As a result, the malicious node has less chance to escape the monitoring mechanism as there are more legitimate nodes in its neighborhood.

The impact of node mobility on the false accusation ratio is presented in Figure 7, in which we can observe a trend in contrast to the previous one: the false accusation ratio continues to increase as nodes move faster. When the maximum speed is 20 m/s, the false accusation ratio is around $5-10\%$. The reason is that higher mobility makes nodes more "memoryless". When nodes are constantly moving at a high speed, a node can overhear only partial information about previous transmissions of its current neighbors. As a result, it is prone to mistakes in cross-checking the incomplete information, and tends to incorrectly accuse its legitimate neighbors. However, even if a single node may have a relatively large chance to do so, the collaborative consensus mechanism can significantly decrease the false accusation ratio by cross-validating the monitoring results from different nodes.

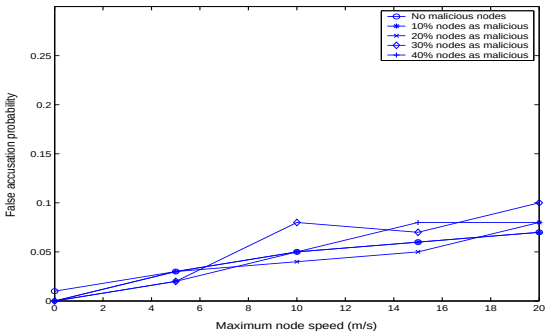

Fig. 7. False accusation probability vs. Mobility

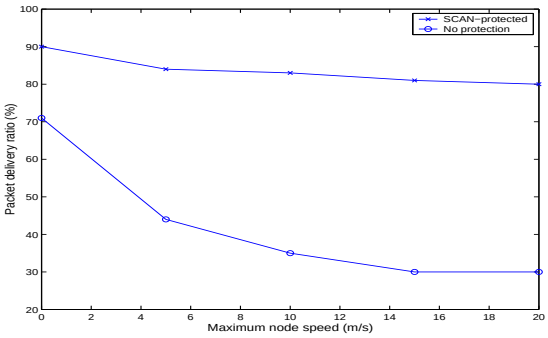

Fig. 8. Packet delivery ratio vs. Mobility

Figure 6 and Figure 7 also illustrate the impact of the number of malicious nodes on the detection performance. We can see that in both cases, even if the number of malicious nodes increases dramatically from 0 to 40% of the network population, it does not exhibit evident impact on the detection performance. One possible reason is that in our simulations, each malicious node acts on its own, and there is no collusion between them. More complicated simulations that take such collusion into account may show different results.

C. Packet Delivery Ratio

The effectiveness of SCAN can be evaluated from the packet delivery ratio perspective. Figure 8 shows the improvement on the packet delivery raio in a SCAN-protected network. In these simulations, 30% of the nodes are set as malicious nodes.

We can see from the figure that SCAN increases the packet delivery ratio by a factor up to 150% even if 30% of nodes are malicious. The reason is that after a malicious node starts to launch the attacks, it is detected by its neighbors and its current token is then revoked. Therefore, it can not participate in the network and disrupt the network operations any more. In an ad hoc network without any security protection, the packet delivery ratio can be as low as 30%, even if the network is lighly loaded as in our simulations. On the contrary, the packet delivery functionality is significantly improved in a SCAN-protected network.

Another observation from Figure 8 is that even in a SCAN-protected and light-loaded network, the packet delivery ratio is

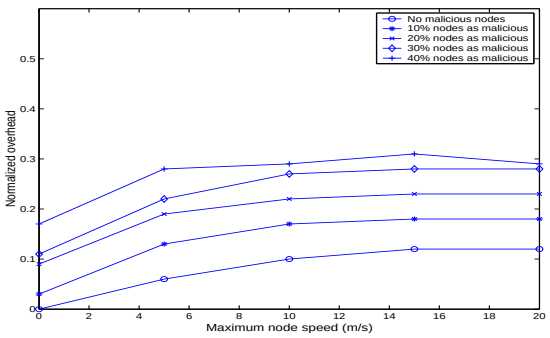

Fig. 9. Communication overhead vs. Mobility

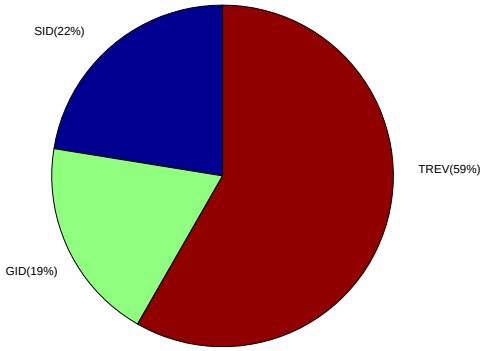

Fig. 10. The distribution of the communication overhead

not 100%. One might suspect the major reason to be the non-zero miss detection ratio, as a few malicious nodes may stay in the network without being detected. However, our traces show that this is not the case. In the simulations, most packet loss is caused during the detection and reaction phases, i.e., after a malicious node has launched attacker yet before it is finally isolated. During this time period, the packet delivery functionality may be adversely affected.

D. Communication Overhead

Lastly we evaluate the communication overhead of SCAN in terms of the total number of SID, GID, and TREV packets sent in the network. Figure 9 shows the normalized overhead under different conditions. We can see that the communication overhead steadily increases as there are more malicious nodes in the network, which is quite intuitive. Furthermore, the communication overhead of SCAN also increases as node mobility increases. This is because when nodes move faster, there is larger chance that a legitimate node is incorrectly suspected or even convicted by its neighbors, as we have shown in Figure 7. As a result, more SID packets and TREV packets are sent, which increases the communication overhead.

Because SCAN may generate three types of packets, namely SID, GID, and TREV packets, we further show the distribution of the communication overhead in Figure 10. From this figure we notice that the dominant portion of the overhead comes from flooding the network with TREV packets to revoke the tokens of convicted malicious nodes. This also explains why the overhead of SCAN steadily increases when there are more malicious nodes in the network.

VII. Discussion

A. Design Rationale

In this section we revisit and elaborate on several design choices in SCAN.

Asymmetric Cryptography Primitive The asymmetric cryptography primitive (RSA) used in SCAN has relatively high computation overhead, compared to the symmetric ones. We justify this design choice by four reasons. First, we pursue a self-organized security design that does not assume any *a priori* secret association between nodes, or the existence of any centralized trusted entity. As a result, one fundamental component in symmetric cryptography based schemes, namely key management, is hard to enforce. Secondly, only tokens and TREV packets are signed in SCAN. We indeed avoid the overhead of performing any cryptographic computation on per routing message basis. Thirdly, the credit strategy in deciding token lifetime further decreases the token renewal overhead over time. Lastly, we believe that with continuous development of hardwares, mobile (even low-end) devices will have more and more computation power in the future.

Localized Intrusion Detection The collaborative monitoring mechanism in SCAN is localized in that each node monitors its neighbors for any misbehavior. This is motivated by the absence of any traffic concentration point in the ad hoc networks. An alternative approach is the end-end-end scheme, in which the sender detects the quality of the route based on the feedback from the receiver. However, the end-to-end approach can only determine whether there is any attacker in the route, instead of which node is the attacker. It also incurs extra communication overhead when the transport layer protocols, such as UDP, and the application layer protocols do not provide any feedback from the receiver to the sender.

Global Intrusion Reaction Our intrusion reaction mechanism guarantees that the attacker is isolated in the network once it is detected by its neighbors. This can be viewed as a global reaction scheme. An alternative approach is the end-to-end reaction scheme, in which a sender tries its best to avoid the attackers that it is aware of. The end-to-end approach is often combined with the end-to-end detection schemes. However, we abandon this approach due to several reasons. First, while it works well with source routing protocols, it is difficult to be extended to work with distance vector routing protocols: once the sender pumps the packets into the network, it cannot control the route along which the packets are forwarded. Secondly, isolation of convicted attackers is also desirable to the network, because the attackers cannot resume the attacks and waste network resource in the future.

B. Related Issues

In this section we comment on several related issues and discuss future work.

Accommodating Other Routing Protocols The monitoring mechanism in SCAN can be easily extended to accommodate other on-demand or even proactive routing protocols than

AODV. The only requirement is that the routing algorithm is distributed and deterministic, so that each node can examine the routing updates advertised by its neighbors, based on the same input to the routing algorithm. SCAN naturally works with source routing protocols such as DSR [10]. Because the entire route is listed in routing updates, a node can directly compare a new update with the previous routes announced by the next hop node therein. DSDV [24] has the same limitation as AODV in that the routing message does not provide the next hop information. Similarly, the addition of a new *next_hop* field in the routing message can address this issue and facilitate the monitoring mechanism.

Colluding Attackers We assume that the collusion among the attackers is limited, i.e., any group of colluding attackers has less than k attackers. More powerful collusion among the attackers will break SCAN as it violates the assumption of the polynomial secret sharing scheme. We will exploit several strategies, for example, re-keying of the SK, multiple SKs for different neighborhoods, to accommodate more general attack model in the future work.

Sybil Attack One potential vulnerability of ad hoc networks is the Sybil attacks [25], in which an adversary may present and abuse multiple entities. As argued in [25], a logically centralized entity that certifies identities is the key factor to thwart Sybil attacks in a distributed system. SCAN does not require any centralized entity physically existed in the network. However, the local neighbors of a newly joined node, which then bootstraps the new node, can jointly serve for the purpose of certifying its identity, since these local nodes can gain some knowledge through physical contact or location-dependent side channels [26].

Node Density SCAN relies on the collaboration among local neighboring nodes in both token renewal and monitoring mechanisms. The effectiveness of SCAN in a sparse network may be significantly affected by the density distribution over the network. However, mobility can help to alleviate this problem as the legitimate nodes roam in the network and collect partial tokens from other nodes met in different neighborhood.

Energy Efficiency SCAN requires that each node turns on its wireless interface and overhears the channel all the time. This may cause significant energy consumption in practice. One possible extension is to make each node periodically wake up and undertake the monitoring responsibility, which trades-off between full strength monitoring and energy efficiency. However, we admit that a more careful study is needed and we leave it for the future research.

VIII. Related Work

The secure ad hoc routing problem has attracted substantial attention in recent years. Hu et al. [4] proposed the Ariadne protocol, which uses TESLA [18] one-way key chains and source-destination pairwise keys to protect the DSR routing protocol. The same authors [5] also proposed the SEAD protocol to secure the DSDV routing protocol based on one-way hash chains. The SRP protocol proposed by Papadimitratos and Haas [6] relied on the secret association between source and destination to protect the source routing messages. Sanzgiri et al. [7] presented the ARAN protocol which exploits asymmetric cryptography to authenticate the routing messages based on each node's public-key certificate, distributed by a central trusted server. The SAODV protocol proposed by Zapata and Asokan [8] uses both one-way hash chains and data signatures to secure the AODV routing protocol.

All these protocols take the proactive approach and prevent malicious attacks by protecting the routing messages through cryptographic primitives. They either assume some kind of *a priori* secret association or key exchange between the nodes, or assume the existence of a centralized trusted server in the network. On the contrary, SCAN takes the reactive approach by detecting and reacting to malicious attacks. SCAN protects the mobile ad hoc networks through self-organized, fully distributed, and localized mechanisms, in which no secret associations exist between a pair of nodes, and no single node is superior to the others. SCAN also differs from these secure routing protocols in that it addresses the protection of routing and packet forwarding in a unified framework.

There have been several papers focused on providing self-organized security support in ad hoc networks. Hubaux et al. [2], [9] proposed a self-organized public-key infrastructure for ad hoc networks, the idea of which was similar to PGP [27]. In this infrastructure, the certificate of each node is issued by other nodes, and the certificate chain is used to verify a given certificate. However, as inherited from the PGP trust model, this design is intolerant of compromised nodes which, unfortunately, are an unavoidable security threat in mobile ad hoc networks. Perhaps the most relevant work to SCAN is the localized certification service proposed by Kong et al. [3]. The token renewal process in SCAN is similar to this scheme. However, SCAN provides a complete network-layer security solution that encompasses all three components of protection, detection, and reaction.

Zhang and Lee [28] were among the first to study the problem of intrusion detection in wireless ad hoc networks. Marti and others [1] proposed *watchdog* that monitors a node based on overhearing the channel. The collaborative monitoring mechanism in SCAN differs from *Watchdog* in two aspects. First, while *Watchdog* focuses on packet forwarding misbehavior, SCAN aims at monitoring both routing and packet forwarding activities of each node. Secondly, SCAN exploits local collaboration to address the inherent imperfectness of the information gathered by channel overhearing. The monitoring result at each individual node does not take effect until its neighbors has reached a consensus. The detection performance is thus significantly improved.

IX. Conclusion

One fundamental challenge for security design in mobile ad hoc networks is the absence of any pre-existing infrastructure support. This work explores a novel self-organized approach to securing such networks. To this end, we have presented SCAN, a network-layer security solution that protects routing and forwarding operations in a unified framework. SCAN exploits localized collaboration to detect and react to security threats. All nodes in a local neighborhood collaboratively monitor

each other and sustain each other, and no single node is superior to the others. The proposed design is self-organized, distributed, and fully localized. Both analysis and simulations results have confirmed the effectivenss and efficiency of SCAN in protecting the network layer in mobile ad hoc networks.

REFERENCES

[1] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," in *Proc. ACM MOBICOM*, 2000.

[2] J. Hubaux, L. Buttyan, and S. Capkun, "The Quest for Security in Mobile Ad Hoc Networks," in *Proc. ACM MobiHoc*, 2001.

[3] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang, "Providing Robust and Ubiquitous Security Support for MANET," in *Proc. IEEE ICNP*, 2001.

[4] Y. Hu, A. Perrig, and D. Johnson, "Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks," in *Proc. ACM MobiCom*, 2002.

[5] Y. Hu, D. Johnson, and A. Perrig, "SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks," in *Proc. IEEE WMCSA*, 2002.

[6] P. Papadimitratos and Z. Haas, "Secure Routing for Mobile Ad Hoc Networks," in *Proc. CNDS*, 2002.

[7] K. Sanzgiri, B. Dahill, B. Levine, C. Shields, and E. Royer, "A Secure Protocol for Ad Hoc Networks," in *Proc. IEEE ICNP*, 2002.

[8] M. Zapata and N. Asokan, "Securing Ad Hoc Routing Protocols," in *Proc. ACM WiSe*, 2002.

[9] S. Capkun, L. Buttyan, and J. Hubaux, "Self-Organized Public-Key Management for Mobile Ad Hoc Networks," *IEEE Transactions on Mobile Computing*, vol. 2, no. 1, January 2003.

[10] D. Johnson, D. Maltz, and J. Jetcheva, *DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Network, Ad Hoc Networking, Chapter 5*, . Addison-Wesley, 2001.

[11] C. Perkins and E. Royer, "Ad Hoc On-demand Distance Vector Routing," in *Proc. IEEE WMCSA*.

[12] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols," in *Proc. ACM MOBICOM*, 1998.

[13] C. Perkins, E. Royer, and S. Das, "Ad Hoc On Demand Distance Vector (AODV) Routin," Internet Draft, draft-ietf-manet-aodv-10.txt, 2002.

[14] S. Das, C. Perkins, and E. Royer., "Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks," in *Proc. IEEE Infocom*, 2003.

[15] IEEE Standard, "Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specifications," 1999.

[16] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level Measurements from an 802.11b Mesh Network," in *Proc. ACM SIGCOMM*, 2004.

[17] Y. Hu, A. Perrig, and D. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks," in *Proc. IEEE Infocom*, 2003.

[18] A. Perrig, R. Canetti, D. Song, and J. Tygar, "Efficient and Secure Source Authentication for Multicast," in *Proc. NDSS*, 2001.

[19] N. Salem, L. Buttyan, J. Hubaux, and M. Jakobsson, "A Charging and Rewarding Scheme for Packet Forwarding in Multi-hop Cellular Networks," in *Proc. ACM MobiHoc*, 2003.

[20] L. Buttyan and J. Hubaux, "Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks," *ACM/Kluwer Mobile Networks and Applications*, vol. 8, no. 5, October 2003.

[21] S. Eidenbenz and L. Anderegg, "Ad hoc-VCG: A Truthful and Cost-Efficient Routing Protocol for Mobile Ad Hoc Networks with Selfish Agents," in *Proc. ACM Mobicom*, 2003.

[22] P. Kyasanur and N. Vaidya, "Detection and Handling of MAC Layer Misbehavior in Wireless Networks," in *Proc. IEEE DSN*, 2003.

[23] J. Yoon, M. Liu, and B. Noble, "Random Waypoint Considered Harmful," in *Proc. IEEE Infocom*.

[24] C. Perkins and P. Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in *Proc. ACM SIGCOMM*, 1994.

[25] J. Douceur, "The Sybil Attack," in *Proc. IPTPS*, 2002.

[26] S. Capkun, J. Hubaux, and L. Buttyan, "Mobility Helps Security in Ad Hoc Networks," in *Proc. ACM MobiHoc*, 2003.

[27] P. Zimmermann, *The official PGP User's Guide*. MIT Press, 1995.

[28] Y. Zhang and W. Lee, "Intrusion Detection in Wireless Ad Hoc Networks," in *Proc. ACM MOBICOM*, 2000.

Hao Yang is currently a Ph.D candidate in the Computer Science Department, University of California, Los Angles (UCLA). He received his B.S. degree from the University of Science and Technology of China in 1998, and his M.S. degree from the Chinese Academy of Sciences in 2001. His research interests include network security, wireless networking, and distributed systems.

James Shu is currently a system/software engineer at Northrop Grumman Corporation. He received the B.Sc. degree in 2001 and the M.Sc. degree in 2003 with specialization in wireless networking from the University of California, Los Angles (UCLA). Previously, he worked at Platinum Technology Inc. as a software programmer developing network security applications. His research interests include wireless security/infrastructure, network protocol, routing algorithm and quality of service.

Xiaoqiao Meng is currently a Ph.D. candidate in the University of California, Los Angeles (UCLA). He received his B.S. degree in control theory from the University of Science and Technology of China in 1998, and his M.S. degree in Pattern Recognition and Intelligent Control from the Institute of Automation, Chinese Academy of Sciences, P.R.China. His research interests include wireless networking, sensor networks and performance evaluation.

Songwu Lu is an associate professor in the Computer Science Department at UCLA. He received both his M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign (UIUC). He received an NSF CAREER award in 2001. His research interests include wireless networking, mobile systems, sensor networks, and wireless network security.