A Cooperative Intrusion Detection System for Ad Hoc Networks

Yi-an Huang
College of Computing
Georgia Institute of Technology
yian@cc.gatech.edu

Wenke Lee
College of Computing
Georgia Institute of Technology
wenke@cc.gatech.edu

ABSTRACT

Mobile ad hoc networking (MANET) has become an exciting and important technology in recent years because of the rapid proliferation of wireless devices. MANETs are highly vulnerable to attacks due to the open medium, dynamically changing network topology, cooperative algorithms, lack of centralized monitoring and management point, and lack of a clear line of defense. In this paper, we report our progress in developing intrusion detection (ID) capabilities for MANET. Building on our prior work on anomaly detection, we investigate how to improve the anomaly detection approach to provide more details on attack types and sources. For several well-known attacks, we can apply a simple rule to identify the attack type when an anomaly is reported. In some cases, these rules can also help identify the attackers. We address the run-time resource constraint problem using a cluster-based detection scheme where periodically a node is elected as the ID agent for a cluster. Compared with the scheme where each node is its own ID agent, this scheme is much more efficient while maintaining the same level of effectiveness. We have conducted extensive experiments using the ns-2 and MobiEmu environments to validate our research.

1. INTRODUCTION

In recent years, with the rapid proliferation of wireless devices, e.g., mobile laptop computers, PDAs, and wireless telephones, the potentials and importance of mobile ad hoc networking have become apparent. A mobile ad hoc network (MANET) is formed by a group of mobile wireless nodes often without the assistance of fixed network infrastructure [20]. The nodes must cooperate by forwarding packets so that nodes beyond radio ranges can communicate with each other. There are a number of important MANET applications, e.g., battlefield operations, emergency rescues, mobile conferencing, home and community networking, and sensor dust [20].

MANETs are much more vulnerable to attacks than wired (traditional) networks due to the open medium, dynamically changing network topology, cooperative algorithms, lack of centralized monitoring and management point, and lack of a clear line of defense. There are recent research efforts, e.g., [29, 10], in securing the ad hoc routing protocols (e.g., [13, 21, 14, 22]). Most of these are *prevention* techniques. Experience in security research in the wired environments has taught us that we need to deploy defense-in-depth or layered security mechanisms because security is a process (or a chain) that is as secure as its weakest link [26]. In addition to prevention, we also need *detection* and *response*, as well as security policies and vulnerability analysis. Although many intrusion detection (ID) techniques have been developed in the wired networks, the vast differences in MANET require that we design new intrusion detection architectures and algorithms.

In this paper, we report our progress in developing ID capabilities for MANET. We first give an overview of the main ideas and results.

First, it is often very hard to distinguish between intrusions and legitimate operations or conditions in MANET because of the dynamically changing topology and volatile physical environment. Intrusion detection thus requires extensive evidence gathering and comprehensive analysis. Building effective ID models requires a systematic approach. In prior work [11], we developed a learning-based algorithm for automatically computing *anomaly detection* models based on the correlations among a large set of features. In this paper, we discuss further how to provide detailed information about intrusions from anomaly detection. We show that, for several (well-known) attacks, we can apply a simple rule to identify the attack type after an anomaly is reported. In some cases, these rules can also be used to identify where the attacking node(s) is(are).

Second, intrusion detection in MANET must be carried out in a distributed fashion because of the absence of infrastructure and fixed topology. In our architecture, a detection agent runs on each "monitoring" node to detect local intrusions, and collaborates with other agents to investigate the source of intrusion and coordinate responses. A MANET node typically has limited battery power, thus it is not efficient to always make each MANET node the monitoring node for itself, especially when the threat level is low. In this paper, we describe a cluster-based detection scheme where a cluster of neighboring MANET nodes can periodically, randomly and fairly elect a monitoring node for the entire neighborhood.

The rest of the paper is organized as follows. In order to put our research into proper context, we first briefly discuss how intrusion detection can complement other security technologies in MANET. We then give an overview of our prior work in anomaly detection and discuss how to apply rules on detecting attack types in Section 3. We present a cluster-based intrusion detection scheme in Section 4. Related work is discussed in Section 5.

2. MOTIVATIONS AND ASSUMPTIONS

Intrusion prevention measures, such as authentication and encryption, e.g., [29, 10, 31, 3], can be used as the first line of defense against attacks in MANET. However, even if these prevention schemes can be implemented perfectly, they still cannot eliminate all attacks, especially the internal or insider attacks. For example, mobile nodes (and their users) can be captured and compromised. The attacker can then obtain the cryptographic keys. There are many other internal attack methods, including using worms and viruses that propagate within MANETs.

Intrusion detection and response presents a second line of defense. Given that new vulnerabilities will continue to be discovered and that our adversaries will continue to invent new attack methods, especially for a relatively new technology such as MANET, we as researchers must focus on developing effective detection approaches. As discussed in Section 1, according to the characteristics of MANET, we need to develop a systematic approach for building detection models, identify attacking source to facilitate response actions, and design a distributed and efficient IDS architecture. Although we currently focus on the ad hoc routing protocols, and different network layers may use different audit data and have different performance and efficiency issues, we believe that in general, the same principles apply to the problems of building ID models for other layers.

It is well understood that for the wired environments we need to deploy defense-in-depth or layered security mechanisms. The same principle applies to MANET because not a single approach can solve all MANET security problems. However, it is not realistic to have all security mechanisms activated at all time because of the resource constraints in MANET. Instead, we need to study the resource consumption characteristics of the security mechanisms, and develop strategies for activating the *appropriate* mechanisms according to run-time requirements. For example, rather than using an always-on cryptography-based prevention technique, periodic intrusion detection (analysis) may be a better strategy to defend against attacks that occur infrequently.

In our current research, we make the following *assumptions*. First, an attacker may try to compromise not only the MANET routing protocols but also the IDS itself. Thus, the IDS must detect problems in its own operations. Second, although our ID algorithms and protocols do not rely on other security mechanisms, authentication (based on cryptography or human interaction) and encryption mechanisms are necessary in many situations in order to respond to attacks (e.g., re-authenticate in order to exclude a compromised node) or raise the security level (e.g., switching to an authenticated protocol if suspicious activities are detected). Third, in most cases, the attacker does not want to take the risk of being detected, especially if effective response actions can take place. Thus, an effective and robust IDS is a good deterrent to attackers. Fourth, we can make intrusion detection efficient because running an ID module only at the clusterhead periodically can still capture sufficient anomaly evidences for many attacks. In short, we are taking an *optimistic* approach in which we develop effective, robust, and efficient ID algorithms and protocols to detect attacks, and invoke other security mechanisms only when necessary (and possible).

3. LOCAL INTRUSION DETECTION TECHNIQUES

In this section, we summarize our prior work in anomaly detection and discuss further how to find out attack types and sources for some known attacks after an anomaly is reported. Let us first describe some attacks in MANET and the intrusions used in our experiments.

3.1 Attacks in MANET

From the point of view of intrusion detection and response, we need to observe and analyze the anomalies due to both the *consequence* and *technique* of an attack. While the consequence gives evidence that an attack has succeeded or is unfolding, the technique can often help identify the attack type and even the identity of the attacker.

Attacks in MANET can be categorized according to their consequences as the following:

1. Blackhole: All traffic are redirected to a specific node, which may not forward any traffic at all.
2. Routing Loop: A loop is introduced in a route path.
3. Network Partition: A connected network is partitioned into k ($k \geq 2$) subnetworks where nodes in different subnetworks cannot communicate even though a route between them actually does exist.
4. Selfishness: A node is not serving as a relay to other nodes.
5. Sleep Deprivation: A node is forced to exhaust its battery power.
6. Denial-of-Service: A node is prevented from receiving and sending data packets to its destinations.

Some of the common attacking techniques are:

1. Cache Poisoning: Information stored in routing tables is either modified, deleted or injected with false information.
2. Fabricated Route Messages: Route messages (route requests, route replies, route errors, etc.) with malicious contents are injected into the network. Specific methods include:

(a) False Source Route: An incorrect route is advertised into the network, e.g., setting the route length to be 1 regardless where the destination is.

(b) Maximum Sequence: Modify the sequence field in control messages to the maximal allowed value. Due to some implementation issues, a few protocol implementation cannot effectively detect and purge these "poluted" messages timely so that they can invalidate all legitimate messages with a sequence number falling into normal ranges for a fairly long time.

3. Rushing: This can be used to improve Fabricated Route Messages. In several routing protocols, some route message types have the property that only the message that arrives first is accepted by a recipient. The attacker simply disseminates a malicious control message quickly to block legitimate messages that arrive later.

4. Wormhole: A tunnel is created between two nodes that can be utilized to secretly transmit packets.

5. Packet dropping: A node drops data packets (conditionally or randomly) that it is supposed to forward.

6. Spoofing: Inject data or control packets with modified source addresses.

7. Malicious Flooding: Deliver unusually large amount of data or control packets to the whole network or some target nodes.

The above lists are by no means complete. Simulating a realistic attack involves selecting a technique(s) that will lead to a consequence(s). This is a non-trivial task. In our experiments, we implement and use the following intrusions:

1. Intrusion I: Blackhole and Sleep Deprivation using False Source Route and Maximum Sequence and Rushing. The attacker broadcasts a falsified source route of length 1 from the victim to any node. The advertisement is rushed and tagged with Maximum Sequence to defeat valid route messages from the victim. The consequences are that the victim node becomes a blackhole and is in sleep deprivation because it receives and forwards all traffic.

2. Intrusion II: Selfishness and Denial-of-Service using Packet Dropping. The malicious node selfishly drops packets and as a result, some node(s) suffers from Denial-of-Service.

3. Intrusion III: Sleep Deprivation using Malicious Flooding. The victim is in sleep deprivation because it is flooded by a large amount of packets.

4. Intrusion IV: Routing Loop using Spoofing. The attacker spoofs some route advertisement messages to create a routing loop.

The logic behind our selection is as follows. They involve most of the attack techniques list above and we can implement under a simulation environment, and they are realistic and relative complete intrusion scenarios each of which includes both one or more attack consequences and attack techniques. Note that although real attack scenarios and intrusion traces are much preferred for research purposes, they are not currently publicly available yet since MANET is still an experimental environment. We look forward to every such possibility and hopefully real data traffic can be integrated into our platform for evaluation soon.

3.2 Anomaly Detection

In our prior work [11], we proposed a learning-based approach for constructing anomaly detection models for MANET routing protocols. We believed that strong feature correlation exists in normal behavior, and that such correlation can be used to detect deviations caused by abnormal (or intrusive) activities. We developed a *cross-feature analysis* anomaly detection approach that explores the correlations between each feature and all other features. This approach computes a classifier C_i for each f_i using $\{f_1, f_2, \ldots, f_{i-1}, f_{i+1}, \ldots, f_L\}$, where $\{f_1, f_2, \ldots f_L\}$ is the feature set. C_i can be learned from a set of training data. It predicts the most likely value of f_i based on the values of other features. The original anomaly detection problem, i.e., whether a record is normal or not, is solved as follows. Given a record $x =< v_1, v_2, \ldots, v_L >$, first apply each C_i to compute $p_i(v_i|v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_L)$, i.e., the probability of v_i given the values of other features. Then compute the *average probability* $\frac{\sum_i p_i}{L}$ and compare it with a threshold. An alarm is raised if it is lower than the threshold because it implies the record is highly unlikely. Obviously, the threshold value determines the false alarm rate. We use a validation data set and a criteria on false alarm rate, e.g., $\leq 1\%$, to set the threshold.

In our study, we define a total of 141 features, which is listed in Appendix A. We did not rely on knowledge of attacks because our goal is to detect anomalies caused by known or new attacks. These features capture the basic view of network topology and routing operations, as well as traffic patterns and statistics. In addition, we include the feature "absolute velocity" which characterizes the physical movement of a node.

Table 1: Experiment Parameters

Parameter	Value/Choice
Classifier	C4.5 [25]
Execution Time (Training)	10000s
Execution Time (Testing)	1000s
Feature Sampling Interval	5s
Node Movement Model	Random Way-Point Model
Peak Movement Speed	$5 \sim 40$m/s
Topology	$500m \times 1500m$
Transmission Range	250m
Maximum Bandwidth	2Mb/s
Maximum Connection	10

We conducted experiments using the ns-2 simulator [9]. For brevity, we only summarize results on one routing protocol, namely, the Ad Hoc On-demand Distance Vector (AODV) protocol [22]. The parameters in our experiments are listed in Table 1. We use trace data of normal runs for training the anomaly detection models. We then run the attacks and collect the trace data for evaluating the models. For example, if in one simulation the total running time is 10,000 seconds, and the sampling rate, by which the feature values are computed, is 5 seconds, then the trace data has 2,000 data points or *events*. Each event is labeled as normal or abnormal according to when and for how long an attack is running (and how long the effect lasts). When evaluating an anomaly detection model, we compute how many abnormal events are correctly identified (i.e., the detection rate) and how many normal events are incorrectly identified as anomalies (i.e., the false alarm rate). In our experiments, an anomaly detection model computes features and detects anomalies on each node locally. As long as there is one detector (on one node) identifies an abnormal event, we count it as a anomaly alert (true detection or false alarm) for the MANET. The intuition is that when there is an intrusion, we need to have at least one node that detects the anomaly. Later in the paper, we will discuss how the MANET nodes can cooperatively identify the attack type and attacker.

Table 2: Experiment Results: Anomaly Detection Only

Attack	Detection Rate	False Alarm Rate
Intrusion I	85%	0.97%
Intrusion II	98%	0.89%
Intrusion III	99%	0.95%
Intrusion IV	87%	0.98%

The detection results are shown in Table 2. Our anomaly detector has better performance on Intrusion II and Intrusion III because their attack techniques are very blatant and the consequences are very obvious. Whereas for Intrusion I and Intrusion IV, the techniques for creating blackhole and routing loop are more subtle, and the consequences are less observable.

3.3 Identifying Attack Types

It is essential that an IDS not only detects an anomaly but also identifies the attack type and the attacker whenever possible. Without them, it is hard to determine how to respond meaningfully without interrupting normal communication. Here we propose an approach to obtain these information after anomalies have been discovered through anomaly detection. The basic idea is to determine the detailed attack information from a set of identification rules, which are pre-computed for known attacks. We are going to show that rules are available for a lot of well-known attacks.

First of all, these rules may involve more features other than those have already been computed and used in anomaly detection. One may point out these rules can be applied in parallel with anomaly detection to save computation time, the extra cost to compute these features may defeat this "optimization" as they are fairly expensive. As a result, they should only be computed after an anomaly is reported, which should be rare.

For each attack, we call the node that runs the corresponding detection rule the "monitoring" node, and the node whose behavior is being analyzed (i.e., the possible attacking or misbehaving node) the "monitored" node. For attacks related to Packet Dropping, the monitoring node is a 1-hop neighborhood of the "monitored" node. Both the attack type and the attacker can be identified because the monitoring node can overhear traffic within its 1-hop neighborhood. For Blackhole attacks, the monitoring node is also the monitored node because the detection rule relies on information that is available only on the node (obviously, if an attacker has full control of the node, then the detection modules can be disabled unless they run on some tamper-resistant device). For Flooding and Maximum Sequence attacks, only the attack type, but not the attacker, can be identified by a monitoring node.

We now describe some notations of statistics (features) used in these rules. We use M to represent the monitoring node and m the monitored node.

- $\#_{(*,m)}$: the number of incoming packets on the monitored node m.
- $\#_{(m,*)}$: the number of outgoing packets from the monitored node m.
- $\#_{([m],*)}$: the number of outgoing packets of which the monitored node m is the source.
- $\#_{(*,[m])}$: the number of incoming packets of which the monitored node m is the destination.
- $\#_{([s],m)}$: the number of incoming packets on m of which node s is the source.
- $\#_{(m,[d])}$: the number of outgoing packets from m of which node d is the destination.
- $\#_{(m,n)}$: the number of outgoing packets from m of which n is the next hop.
- $\#_{([s],M,m)}$, the number of packets that are originated from s and transmitted from M to m.
- $\#_{([s],M,[m])}$, the number of packets that are originated from s and transmitted from M to m, of which m is the final destination.
- $\#_{([s],[d])}$, the number of packets received on the monitored node (m) which is originated from s and destined to d.

These statistics are computed over a feature sampling interval, denoted as L_s. In addition, we often need the same set of statistics that are computed over a longer period. These longer-term statistics can be computed directly from basic features by aggregating them in multiple feature sampling intervals. We use FEATURE^L to denote the aggregated FEATURE over a long period L. We always assume that time interval L is multiples of L_s, for simplicity. For example, the notion $\#^L_{(*,m)}$ are computed by summing up all $\#_{(*,m)}$ in L/L_s rounds of feature sampling intervals.

We also need finer-grained statistics on specific types of packets, e.g., the number of certain route control messages.

These specific statistics are denoted by appending a predicate to the corresponding feature. For instance, $\#_{(*,m)}$(TYPE=RREQ) represents the number of incoming RREQ (route request) packets on the monitored node m.

Below we describe the identification rules for several well-known attacks.

Unconditional Packet Dropping Monitor the statistics FP (Forward Percentage)

$$\mathrm{FP}_m = \frac{\text{packets actually forwarded}}{\text{packets to be forwarded}} = \frac{\#^L_{(m,M)} - \#^L_{([m],M)}}{\#^L_{(M,m)} - \#^L_{(M,[m])}}$$

over a sufficiently long time period L. FP determines the ratio of forwarded packets over the packets that are transmitted from M to m and that m should forward. If the denominator is not zero and $FP_i = 0$, the attack is detected as Unconditional Packet Dropping and m is identified as the attacker.

Random Packet Dropping Monitor the same statistics FP as Unconditional Packet Dropping. If the denominator is not zero and FP_m is less than a chosen threshold $\epsilon_F P$ ($\epsilon_{FP} < 1$) but not zero, the attack is detected as Random Packet Dropping and node m is identified as the attacker. Note that we cannot use $\epsilon_F P = 1$ since there is a small chance that packets are dropped due to some topology and buffering reasons (overloaded buffer, routes no longer valid, etc.) Therefore, ϵ_{FP} is chosen so that $1 - \epsilon_{FP}$ is equal to the upper bound of dropping rate that can be tolerated.

Selective (Random) Packet Dropping Monitor the statistics LFP (Local Forward Percentage)

$$\begin{aligned}\mathrm{LFP}^s_m &= \frac{\text{packets from source } s \text{ actually being forwarded}}{\text{packets from source } s \text{ to be forwarded}} \\ &= \frac{\#^L_{([s],m,M)}}{\#^L_{([s],M,m)} - \#^L_{([s],M,[m])}}\end{aligned}$$

over a sufficiently long time period L for each source s. If the denominator is not zero and the statistics is zero (unconditional dropping), the attack is unconditional Packet Dropping targeted at s. Likewise, if the LFP is less than ϵ_{LFP} ($\epsilon_{LFP} < 1$), the attack is random Packet Dropping targeted at s. In either case, m is identified as the attacker.

Blackhole Monitor the statistics GFP (Global Forward Percentage)

$$\mathrm{GFP}_M = \frac{\#^L_{(*,M)} - \#^L_{(*,[M])}}{\sum\limits_{i \in N(M)} \#^L_{(i,M)} - \sum\limits_{i,j \in N(M)} \#^L_{(i,[j])} - \#^L_{(*,[M])}}$$

over a time period of L. The numerator is the total number of packets that are received by M and M should forward. The denominator is the total number of packets sent by M's 1-hop neighborhood ($N(M)$) and are not destined for another neighbor or M. If all such packets are being absorbed by M for a sufficiently long period, or more precisely, if the denominator is not zero and GFP = 1, then an blackhole is detected and M is identified as the attacking or misbehaving node. Note that the statistics must be collected on M locally. The detection of blackhole may be infeasible if M is malicious and the attacker has total control of M so that the detection modules can be disabled. However, in some other situations, such as Intrusion I, where the blackhole is actually a passive victim of sleep deprivation attack, the node can use this rule to detect the blackhole condition and initiate appropriate response actions.

Malicious Flooding on specific target Monitor the total number of $\#^L_{([m],[d])}$ over a period of time L for every destination d. If it is larger than threshold $MaxCount$ (which should be a system parameter based on the upper bound of normal traffic volume), the attack is a Malicious Flooding.

Maximum Sequence Monitor MSC (Maximum Sequence Counter)

$$\mathrm{MSC(TYPE}{=}t)_s = \#^L_{([s],*)}(\mathrm{TYPE}{=}t,\ \mathrm{SEQ{=}MAX\_SEQ})$$

for every source s and message type t over the period of L. MAX_SEQ is specific to each routing protocol. Typically, if the field is 32-bit long, the maximal value is $2^{32} - 1 = 4294967295$. If MSC for any source s is not zero, the attack is detected. Note that an alarm can be triggered immediately as long as the condition holds, even before L has elapsed. There is a small chance that the maximum sequence number is used in normal processing. However, the chance is so small (even if 1000 control message is sent per second, 50 days are needed before the maximal sequence number is reached) that a false alarm under this case can perhaps be tolerated. Note that our implemention currently can only detect an attack with the unique sequence number MAX_SEQ. A smarter variation of this attack can be implemented in such a way that a fairly large sequence number, but not MAX_SEQ, is used. A even much smarter one may involve using a series of "large" sequence numbers that are continuously changed over time, in order to evade detections. To detect them, we need to have a self-adjusting non-determinstic rule that can detect "unusually" large sequence values, based on historical statistics. The work, as well as other rules to detect a series of more complicated types of attacks, is still in progress.

In our experiments, all period L is set to be $5 \times L_s$ and L_s is 5 seconds. We set ϵ_{FP} and ϵ_{LFP} to be 0.90. The $MaxCount$ used by Malicious Flooding detector is set to be 1000 packets.

For the Packet Dropping and Blackhole attacks, the above rules can identify not only the attack type but also the attacking or misbehaving node because the monitoring node can overhear traffic within its 1-hop neighborhood. For the other two attacks, the attacking node cannot be reliably identified because packets from remote nodes may be spoofed. In general, attacker identification is a very hard problem and remains an open research issue.

These identification rules is activated only when some anomaly has been observed by the anomaly detection model. First of all, If an alert is produced by the anomaly detection model, it is labeled with as UNKNOWN. The alert is then relabeled with the corresponding attack type(s) if an identification rule applies. Otherwise, it remains to be UNKNOWN.

We show the performance of the new scheme in Table 3. Here, the definition of detection rate is changed slightly due to the use of attack type labels. Detection rate is the percentage of attacks detected and labeled correctly by the misuse detection model. Partial detection rate is the percentage of attacks detected and labeled as UNKNOWN, i.e., those detected by the anomaly detection model only. Misclassification rate is the percentage of attacks that are labeled incorrectly by the misuse detection model. We call the sum of these three measures the overall detection rate. Finally, we show the false alarm rate.

Table 3: Experiment Results with Identification Rules, where DR=Detection Rate, PDR=Partial Detection Rate, MR=Mis-classfication rate, FAR=False Alarm Rate

Attack	DR	PDR	MR	FAR
Intrusion I	78%	7%	0%	1%
Intrusion II	91%	6%	1%	1%
Intrusion III	98%	0%	1%	1%
Intrusion IV	-	87%	0%	1%

We can see that in the new scheme, the overall detection rate, i.e., the sum of the first three columns, is always the same as the detection rate of anomaly detection model alone (see Table 2). This is not surprising because the rules are used to (further) identify the attack type only after an anomaly is detected. The overall detection rate is not changed because no additional attacks will be detected. We can see that most of the well-known attacks have been detected.

4. CLUSTER-BASED INTRUSION DETECTION

So far in the paper we have assumed that each MANET node is a monitoring node that runs some ID models. MANET nodes typically have limited battery power, thus it is not efficient to make each MANET node always a monitoring node, especially when the threat level is low. Instead, a cluster of neighboring MANET nodes can randomly and fairly elect a monitoring node, the clusterhead, for the entire neighborhood. In other words, the responsibility of intrusion detection is shared among nodes in the cluster. In this section, we present cluster formation algorithms and cluster-based intrusion detection schemes.

4.1 Cluster Formation Protocols

A MANET can be organized into a number of clusters in such a way that every node is a member of at least one cluster. A cluster is defined as a group of nodes that are *close* to each other. The criteria of 'close' is that a node in the cluster, the *clusterhead*, has all other members, known as *citizens*, in its 1-hop vicinity. As a special case, a node that cannot be reached by anyone else (or under other special circumstances as described below) forms a *single node cluster*, or SNC. The *size* of a cluster is defined as the number of nodes in the cluster (including both clusterhead and citizens) and is denoted as S_C.

It is imperative that clusterhead assignment be fair and secure. By fairness, we mean that every node should have a fair chance to serve as a clusterhead. Note that fairness has two components, *fair election*, and *equal service time*. We currently do not consider differentiated capability and preference (such as criteria based on network or CPU load, unless they can be verifiable) and assume that every node is equally eligible. Thus, *fair election* implies randomness in election decision, while *equal service time* can be implemented by periodical fair re-election. By security, we mean that none of the nodes can manipulate the selection process to increase (or decrease) the chance for it (or another node) to be selected. Obviously, if randomness of the election process can be guaranteed, then security is also guaranteed.

Although there are other cluster formation protocols available, they do not satisfy our requirements discussed above. For example, the Leader Election (or cluster organization) algorithms in [28, 2] choose either a common evaluation function or node-specific utility functions to compute a score for every node, and the node with maximal score is elected as the clusterhead. They do not guarantee the random selection of clusterheads. They may allow a node to advertise a high score for itself, unless care is taken to make the evaluation process verifiable and with non-repudiation.

We use several techniques to guarantee the fairness and security of the election process. Firstly, each node i contributes a random value R_i to the input. Then a common selection function is used by all nodes to compute a integer from 0 to $S_C - 1$ from a total of S_C inputs. The output of the election function must have a uniform distribution in [0, $S_C - 1$]. The selection function we use is simply the modular Exclusive OR (or XOR) function, i.e., $f(R_0, R_1, R_2, ..., R_{S_C-1}) = (\bigoplus_{i=0}^{S_C-1} R_i)$ MOD S_C. A nice property of XOR is that as long as one input is random (i.e., from a "well-behaving" node), the output is random. The random values are fully exchanged within the cluster (clique) and the selection function is computed in a distributed manner, i.e., on each node, to decide the clusterhead. This guarantees that the same clusterhead be computed by all cluster members.

Before we describe our clustering formation protocols, we state a few assumptions about the MANET environment.

- Each node contains a unique and ordered identifier.
- All links are bidirectional.
- Every node can overhear traffic within its transmission range (this is a common requirement by MANET monitoring schemes, e.g. [18]).
- Neighbor information is always available. Usually this is implemented by periodically broadcasting HELLO messages and listening to the neighbors' response. Given the assumption, we can obtain the number of neighbors of node i. Let us denote the value to be N_i.
- A secure, fast and reliable node to node communication infrastructure is available. The infrastructure has to be light-weighted because MANET nodes are often resource-constrained. Recently, efficient protocols for MANET are proposed, such as TESLA [23], which carries only symmetric cryptographic functions. These protocols make certain assumption that loose synchronized clocks are available.

Figure 1 shows a finite state machine demonstrating the states of the MANET nodes and the state transitions that are enabled by the cluster formation protocols.

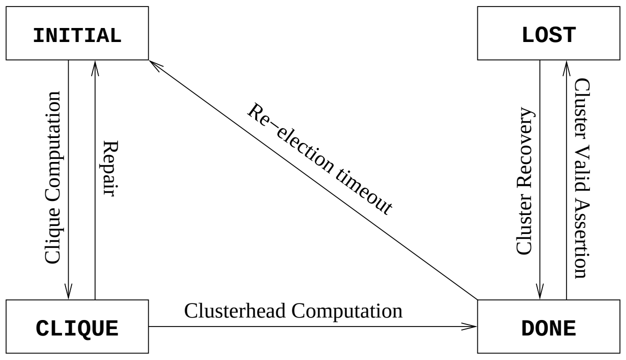

Figure 1: Finite State Machine of the cluster formation protocols

Initially, all nodes are in an INITIAL state. They temporarily assume themselves SNCs, so that they can do intrusion detection for themselves, just as in the per-node based approach. We perform an initial clusterhead setup round, which is composed with two protocols: *Clique Computation* and *Clusterhead Computation*.

Clique Computation Protocol A clique is defined as a group of nodes where every pair of members can communicate via a direct wireless link. Note that the definition of a *clique* is stricter than the definition of a *cluster*. The clique requirement can be relaxed right after the clusterhead has been computed. That is, only the clusterhead needs to have direct links with all members. We use the cluster formation algorithm from [15] to compute cliques. Once the protocol is finished, every node is aware of its fellow clique members. We denote the clique containing i as CL_i, i.e., $\forall j \in CL_i, CL_j = CL_i$. We define $CL'_i = CL_i - \{i\}$.

Once the Clique Computation Protocol has finished, all nodes enter CLIQUE state.

Clusterhead Computation Protocol The purpose of this protocol is to randomly select one node in the clique as the clusterhead. Without loss of generality, we discuss the behavior on the i-th node.

1. Generate a random integer R_i.
2. Broadcast a message ELECTION_START=(ID_i, HASH(ID_i, R_i)) to CL'_i. HASH is a common hash function. A corresponding timer T_1 is setup.
3. On Receiving *all* ELECTION_START from CL'_i, broadcast the message ELECTION=(ID_i, R_i) to clique CL'_i.
4. If T_1 is timeout, every node for whom ELECTION_START has not be received is excluded from CL_i.
5. On Receiving ELECTION from node j, verify its hash value matches the value in the ELECTION_START message from j. Store R_j locally.
6. If all R_j from CL'_i have arrived, compute H=SEL(R_0, $R_1, R_2, ..., R_{S_C-1}$) where SEL is the selection function. Determine the clusterhead H as the h-th node in the clique since all IDs are ordered.
7. If $H \neq i$ (i.e., as a citizen), do the following.
   (a) Send ELECTION_DONE to H.
   (b) Wait for ELECTION_REPLY from H, then enter DONE state.
8. Otherwise, as a clusterhead, H performs following.
   (a) Setup a timer T_2.
   (b) On Receiving ELECTION_DONE, verify it is from CL'_i.
   (c) If T_2 is timeout, citizens from whom ELECTION_DONE has not be received are excluded from CL_i. Broadcast ELECTION_REPLY to CL'_i and enter DONE state.

Once the clusterhead is determined, it copies the clique member list to a citizen list CT_C. The suffix C denotes the current cluster controlled by the clusterhead.

Cluster Valid Assertion Protocol All nodes should perform this assertion periodically in DONE state. There are two parts in this protocol.

1. Since the network topology tends to change in an ad hoc network, connections between the elected clusterhead and some citizens nodes may be broken from time to time. If a link between a citizen Z and a clusterhead H has been broken, Z will check if it is in another cluster. If not, it enters LOST state and activates the *Cluster Recovery Protocol*. Also, Z is removed from H's citizen list CT_C. If there is no more citizens in cluster C, H becomes a citizen if it belongs to another cluster. Otherwise, H enters LOST state and activates the *Cluster Recovery Protocol*.
2. Even if no membership change has occurred, the clusterhead cannot function forever because it is neither fair in terms of service and unsafe in terms of the long time single-point control and monitoring. We enforce a mandatory re-election timeout, T_r. Once the T_r expires, all nodes in the cluster enters the INITIAL state and start a new clusterhead setup round. If the clique property still holds, the *Clique Computation* step can be skipped.

Cluster Recovery Protocol In the case that a citizen loses its connection with previous clusterhead or a clusterhead loses all its citizens, it enters LOST state and initiate *Cluster Recovery Protocol* to re-discover a new clusterhead. Again, without loss of generality, we discuss the behavior on the i-th node.

1. A request message ADD_REQUEST=(ID_i) is broadcast with a timer T_3.

2. A clusterhead H receives the request and replies ADD_REPLY=(ID_H) only after a short delay T_d (0.5s in our implementation). The delay is introduced in hope that a connection has been stable for T_d can remain to be stable for a fairly long time.

3. Node i replies the first ADD_REPLY it received, i.e., ADD_ACK=(ID_i). And enters DONE state. Additional ADD_REPLYs are ignored.

4. On Receiving ADD_ACK, H adds i into its CT_C.

5. If T_3 is timeout and no ADD_REPLY is received, there is no active clusterhead nearby. Node i enters INITIAL state to wait for other lost citizens to form new cliques and elect their new clusterheads.

4.1.1 Discussion

Since clusters can overlap, a node can belong to multiple clusters. Therefore, the notation CL_i of node i can actually take multiple values. For simplicity of the protocol description, we use the singular form but keep in mind that a node in mulitple clusters should perform the Clusterhead Computation Protocol for each of its clusters independently.

In the Clusterhead Computation Protocol, we assume the topology remains static during computation. In a mobile environment, this assumption does not always hold. A remedy is for each cluster member to monitor the neighborhood actively. Once a link is broken, a REPAIR message is broadcast by both ends of the link, and all other nodes in the cluster will be aware of that. All nodes in the cluster then re-enter INITIAL state and restart the protocol *Clique Computation*.

We require that nodes have direct links to each other (i.e., they are in a clique) in the cluster formation process. This is intentional so that spoofed messages can be detected and contested because the nodes can overhear each other. Whenever such dispute arises, the nodes can switch to a more secure way (e.g., authenticated) to exchange messages.

Finally, our protocols are meant to be a framework that can be customized according to operational conditions and security needs. For example, a malicious node has a $\frac{1}{S_C}$ chance to be elected as the clusterhead. It can then launch certain attacks without being detected because it is the only node in the neighborhood that is supposed to run the IDS and its IDS may have been disabled already. If this chance is not acceptable, multiple rounds of clusterhead computation can be used to elect multiple clusterheads, each running a separate IDS to monitor the cluster. The extreme is to run an IDS on each node. It is obvious that there is a tradeoff between efficiency and security. We plan to further investigate how to dynamically adjust the number of clusterheads (or monitoring nodes) according to resource constraints and potential threats.

4.1.2 Security Concerns

As an approach dedicated to detect malicious behavior, our protocol itself has to be secure as well. In addition to conventional attacks such as man-in-the-middle and replay attacks (which are addressed by enforcing a secure communication channel and sequence numbers verifiable by neighbors), we have also addressed following specialized attacks with special consideration.

- **Defending against delayed random value distribution:** In order to prevent a malicious node from manipulating the election outcome, e.g., by sending its random number only after it receives the random numbers from all other nodes, the exchange of random numbers among the nodes proceeds in two rounds. First, each node computes a random number and its hash using a common hash function, then sends out only the hash value. Second, only after receiving all hash values from all other nodes, a node sends out the actual random number. A multi-round process for exchanging the random numbers, which corresponds to Steps 1 through 4 in the Clusterhead Computation Protocol, is used to prevent cheating.

- **Defending against intentional timeout for certain advantages:** In the *Cluster Recovery Protocol*, the new member will not have a chance to be elected as a clusterhead in the beginning unless a new re-election period occurs (or if the clusterhead leaves the area). This is intentional so as to reduce the chance that change of clusterhead occurs too often. However, the property involves a fairness issue. A node can refuse to acknowledge being elected as a clusterhead in the cluster computation stages but later on dispatches an ADD_REQUEST to join the cluster. In this way, it will be exempted from serving as a clusterhead (a special type of Denial-of-Service). A similar attack works in the opposite way. An attacker can refuse to take the responsiblity as a citizen (or non-clusterhead member) by repeated timeout until the compromised node is elected as a clusterhead. This gives the attacker the advantage of a clustehead but not willing to conform to a citizen's liability when other nodes are clusterheads. To defeat both of these attacks, we add a retreating suspicion counter in the cluster computation protocol which counts how many times an elected node refuses to respond. If it happens more than certain times (three in our experiments), the node is excluded from further clusterhead computation and an exception is reported about the misbehavior of that node.

4.2 Detection Schemes

Using the cluster formation protocols described above, a clusterhead is selected to perform IDS functions for the whole cluster. It instructs the cluster citizens on how the feature computation is to take place. Note that every node is still required to act as a clusterhead sometime, thus it is still necessary to have the trained IDS models pre-installed on all nodes.

There are several schemes on how and where features are computed and transmitted.

LFSS (Local Feature Set Scheme) For each feature sampling period, a randomly selected cluster member (which can be the clusterhead itself) is requested to transmit its whole feature set to the clusterhead.

CLFSS (Clusterhead-Assisted Local Feature Set Scheme) This is a modified version of LFSS. In LFSS, feature computation is still done on citizens nodes. In order to reduce the burdens on citizen nodes and the traffic overhead, the clusterhead can help compute some of the features, more specifically, traffic-related features (please refer to the appendix for the details of these features). In general, the clusterhead overhears incoming and outgoing traffic on all members of the cluster. Corresponding traffic-related features are computed in the same way as a local node does, as if the cluster is a large node as a whole, where, nevertheless, internal packets (from one cluster member to another) *do* exist and should count. Traffic-related features involving packet counts (66 out of 141 features totally) are normalized by averaging on the cluster size. The citizens (more precisely, one citizen at a time according to the LFSS scheme) are still responsible for computing and transmitting route and location related features to the clusterhead. The clusterhead then evaluates the IDS model based on the revised feature set. We note that it takes significant computational effort to obtain these traffic-related features. Therefore, the overall cluster-wise feature computation costs can be significantly reduced because the vast majority of features are traffic-related features.

4.2.1 Results

We conducted experiments with different density and movement rates using ns-2 and the MobiEmu [30] software, which emulates a wireless ad hoc network on a local area network and thus enabling application-level programs and the measurement of CPU usages. The following criteria are then measured.

1. **CPU Speed-up:** How much of the total CPU usage is reduced in comparison with the per-node based IDS approach? The total CPU usage of a cluster-based IDS includes the amount of CPU cycles in all nodes of the cluster. The three factors consuming the CPU cycles are 1) cluster formation protocols; 2) feature computation; and 3) IDS computation (on the clusterhead only). Whereas for the per-node based IDS, the total CPU usage includes feature computation and IDS computation on all nodes.

2. **Network Overhead:** How much overhead does the communication in the cluster formation protocol and feature exchanges between the clusterhead and the citizens add to normal traffic?

3. **Accuracy:** How does the cluster-based detection schemes perform in terms of accuracy? Is it better or worse compared with the original per-node based algorithm?

The results of CPU Speed-up, in terms of $\frac{\text{CPU cycles in per-node based approach}}{\text{CPU cycles in cluster-based approach}}$, and network overhead, in term of kilobytes transmitted in the whole network, are shown in Figures 2 and 3, respectively. We measure CPU usage by executing the UNIX command "time". Results on detection accuracy for Intrusion I (Blackhole) are shown in Figure 4, in terms of detection rate where the false alarm rate is 1%. Detection results for other three intrusions are not shown due to space limit, but they have similar patterns as Figure 4.

We can see that CLFSS is significantly superior in terms of CPU usage and network overhead than LFSS. Its detection accuracy is around 84%, which is just a little worse than LFSS and a little worse than the per-node based anomaly detection scheme (87%, see Table 2). The accuracy figure shows that the performance of CLFSS is more sensitive to mobility than the LFSS and the per-node based scheme. This is a natural result since we use aggregated traffic-related features, but route and location related features still involve one of the cluster members only at each period. In a highly dynamic scenario, it is more likely the correlation among these patterns are not as regular as that in a per-node based scheme. However, even under a resonably high mobility level (40m/s), the detection rate is still higher than 80%. Considering the significant performance benefit from both CPU usage and network overhead, it is clear that overall, CLFSS is a far better approach than the others.

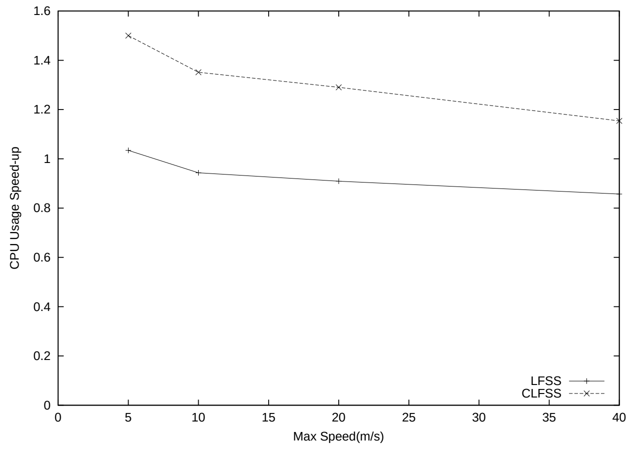

Figure 2: CPU Speed-up vs. Mobility

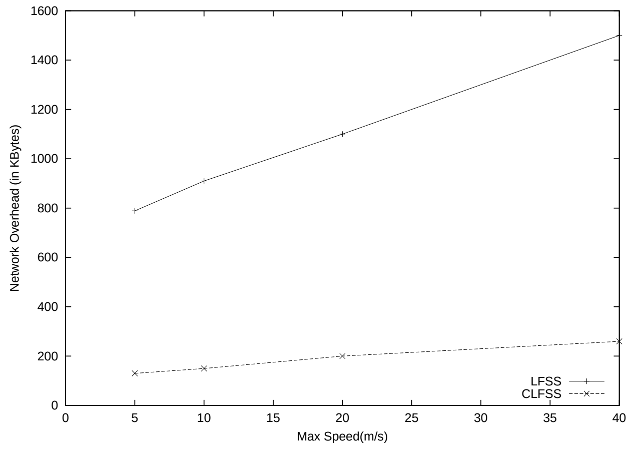

Figure 3: Network Overhead (in KBytes) vs. Mobility

5. RELATED WORK

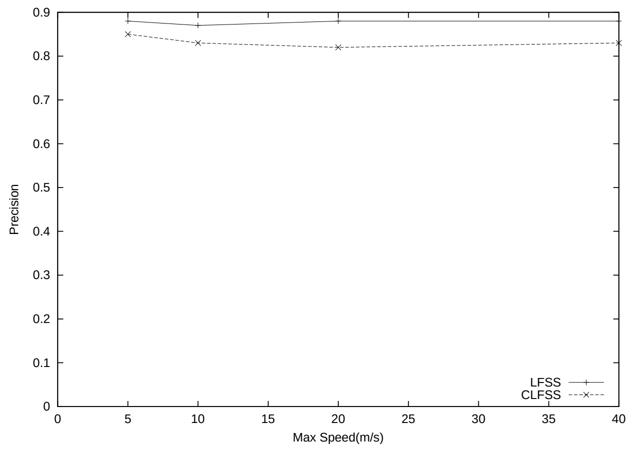

Figure 4: Detection Rate vs. Mobility

Hierarchial network is an effective way to group (or cluster) a large number of nodes in a network. Distributed algorithms to form clusters have been studied extensively, e.g., [2, 15, 28]. Most of these approaches have the drawback that the clusterhead computation can be easily manipulated (cheated) to elect an arbitrary compromised node. Nevertheless, the head-less cluster formation and maintenance protocol [15] does not have such problem. We use this cluster formation scheme as the basis of clique computation protocol because the clique structure allows us to effectively compute a selection function on random inputs from each member.

Although there are secure routing approaches in wired networks, such as [6, 27], they usually come with large communication overhead and do not work well in MANET because of its dynamically changing network topology. Several researchers have recently proposed new secure routing protocols or security enhancement for existing protocols for MANET. Zapata [29] proposes the use of asymmetric cryptography to secure the AODV protocol [22]. Hu et al. [10] consider the problem of avoiding expensive public key computation in authentication in Ariadne, a secure version of the DSR protocol [13]. It primarily uses TESLA [23] an efficient broadcast authentication protocol that requires loose time synchronization, to secure route discovery and maintenance. To use TELSA for authentication, a sender generates a hash chain and determines a schedule to publish the keys of the hash chain. The key required to authenticate a packet will not be published before the packet has arrived at the receiver so that an adversary cannot have captured the key and forged the packet. This is the same underlying broadcast authentication protocol that we currently use.

Even if the above prevention schemes are perfect and implemented correctly, there are still internal and insider attacks that utilizes software vulnerability (e.g., viruses and worms) and social engineering. A compromised node is an insider, with all the necessary cryptographic keys, and can launch many attacks. Thus, intrusion detection is still needed as a second line of defense.

As to intrusion detection in wired environments, since its early introduction [1, 8], it has received increasing interests from researchers and even vendors. The representative misuse detection systems are IDIOT [17] and STAT [12], which use Colored Petri Nets and State Transition Diagrams, respectively, to represent and pattern-match known intrusions.

In protection of routing protocols, Mittal and Giovanni [19] suggests the use of sensors present on links to utilize topology information to detect routing-based attacks. A MANET cannot learn topology information in advance, thus, this technique cannot be directly applied to wireless networks. Cheung and Levitt [7] proposed a detection-response approach to network monitoring. Their scheme also requires topology information to predict the expected behavior. Thus, it is not practical in MANET. Wu et al. [24] proposed an approach to protect OSPF protocol using statistical anomaly detection. Although their work is specific to link-state protocols, the basic principle to detect intrusions with anomaly detection applies to wireless networks as well.

Researchers have begun to investigate detection and response schemes for MANET.

SPARTA, suggested by Krugel et al. [16], builds IDS based on mobile agents. It also features an event definition language (EDL), which describes multiple-step correlated attacks from an intrusion specification database. However, we have not seen details on how these specifications are generated and used for well-known routing attacks.

Watchdog and pathrater approach, discussed by Marti et al. [18], introduces two related techniques to detect and isolate misbehaving nodes, which are nodes that do not forward packets. In the "watchdog" approach, a node forwarding a packet verifies the next hop also forwards it. If not, a failure tally is incremented and misbehavior will be recognized if the tally exceeds certain threshold. The "pathrater" module then utilizes this knowledge of misbehaving nodes to avoid them in path selection. The approach is limited in several aspects. First of all, overhearing does not always work in case of collisions or weak signals. Secondly, pathrater actually awards the misbehaving node, if its motivation comes from selfishness, i.e., not "serving" others can reduce its battery power consumption. It does not prevent the misbehaving node from sending or receiving its own packets.

CONFIDANT [4] extends Marti's approach in numerous ways. Misbehaving nodes are not only excluded from forwarding routes, but also from requesting their own routes. Also, it includes a trust manager to evaluate the level of trust of alert reports and a reputation system to rate each node. Only reports from trusted sources are processed. However, trust management in MANETs has not been well studied yet. For example, it is not clear how fast the trust level can be adjusted for a compromised node, especially if it has a high trust level initially.

Buttyan et al. [5] suggests the use of tamper-resistant hardware on each node to encourage cooperation. Nodes are assumed to be unwilling to forward packets, unless it is stimulated. In this approach, a protected credit counter runs on the tamper-resistant device. It increases by one when a packet is forwarded. It refuses to send its own packets if the counter is smaller than a threshold n. Public key technology is used to exchange credit counter information among neighbors and verify if forwarding is really successful. The scheme has a few strong assumptions, including tamper-resistant hardware and public key technology, which may not be widely available in MANET.

6. CONCLUSION

In this paper, we have reported our progress in developing intrusion detection techniques for MANET. Building on our prior work on local anomaly detection, we further investigated how to provide more accurate information on attack types when an anomaly is found. We have presented a set of rules that can identify the attack type of several well-known attacks. In some cases the rules can also identify the attacking or misbehaving nodes. Our experiments showed that these rules are highly accurate on identifying the corresponding attacks.

In order to address the run-time resource constraint problem, we have developed a cluster-based detection approach. The idea is to elect a node, the clusterhead, to perform IDS functions for all nodes within a cluster. We presented cluster formation protocols that achieve fairness and security in clusterhead election. We evaluated two feature computation schemes. Our experiment results showed that, when the MANET mobility is low, the best scheme reduces host CPU utilization by up to 29% while maintaining the same level of detection performance as the original per-node detection scheme.

6.1 Future Work

We need to guarantee that the IDS cannot be compromised, or at the least, attacks against the IDS can be detected. This is particularly important when using the cluster-based detection approach. If a compromised node happens to be elected as the clusterhead, it can launch attacks without being detected because it is the only node that should run an IDS and its IDS may have been disabled already. A solution may be to use a tamper-resistant device to protect the IDS on each node. We will further investigate this issue.

Attacker identification is another important issue. For some known attacks, the attacking or misbehaving node can be identified by its neighbors. However, the general problem is very challenging. This problem can be partially solved by enforcing authentication on all packets. However, some routing related attacks can still be launched without the need to hide or spoof because by the time an anomaly is reported, it may already be too late to correlate the anomalous behavior with packets received before. We will build on our current work and develop a better and more general solution.

7. REFERENCES

[1] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, April 1980.

[2] S. Basagni. Distributed clustering for ad hoc networks. In *ISPAN-99, International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 310–315, Perth, Western Australia, June 1999.

[3] S. Basagni, K. Herrin, D. Bruschi, and E. Rosti. Secure pebblenets. In *Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, Long Beach, CA, October 2001.

[4] S. Buchegger and J. L. Boudec. Performance analysis of the CONFIDANT protocol: Cooperation of nodes — fairness in dynamic ad-hoc networks. In *Proceedings of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Lausanne, CH, June 2002. IEEE.

[5] L. Buttyan and J. P. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM Journal for Mobile Networks (MONET), special issue on Mobile Ad Hoc Networks*, 2002.

[6] S. Cheung. An efficient message authentication scheme for link state routing. In *Proceedings of the 13th Annual Computer Security Applications Conference*, 1997.

[7] S. Cheung and K. Levitt. Protecting routing infrastructures from denial of service using cooperative intrusion detection. In *New Security Paradigms Workshop*, 1997.

[8] D. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, 13(2), February 1987.

[9] K. Fall and e Varadhan. *The ns Manual (formerly ns Notes and Documentation)*, 2000. Online reference: http://www.isi.edu/nsnam/ns/ns-documentation.html.

[10] Y. Hu, A. Perrig, and D. B. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom 2002)*, September 2002.

[11] Y. Huang, W. Fan, W. Lee, and P. Yu. Cross-feature analysis for detecting ad-hoc routing anomalies. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, Providence, RI, May 2003.

[12] K. Ilgun, R. A. Kemmerer, and P. A. Porras. State transition analysis: A rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[13] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[14] Y.-B. Ko and N. H. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. *ACM/Baltzer Wireless Networks (WINET) journal*, Vol 6-4 - Extended version of the Mobicom'98 paper., 2000.

[15] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM Computer Communication Review*, 27(2):49–64, 1997.

[16] C. Krugel and T. Toth. Flexible, mobile agent based intrsuion detection for dynamic networks. In *European Wireless*, 2002.

[17] S. Kumar and E. H. Spafford. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Information Security Conference*, pages 194–204, 1995.

[18] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Mobile Computing and Networking*, pages 255–265, 2000.

[19] V. Mittal and G. Vigna. Sensor-based intrusion detection for intra-domain distance-vector routing. In R. Sandhu, editor, *Proceedings of the ACM Conference on Computer and Communication Security (CCS'02)*, Washington, DC, November 2002. ACM Press.

[20] C. E. Perkins. Ad hoc networking: An introduction. In C. E. Perkins, editor, *Ad Hoc Networking*. Addison-Wesley, 2000.

[21] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[22] C. E. Perkins and E. M. Royer. The ad hoc on-demand distance-vector protocol. In C. E. Perkins, editor, *Ad Hoc Networking*. Addison-Wesley, 2000.

[23] A. Perrig, R. Canetti, D. Tygar, and D. Song. The TESLA broadcast authentication protocol. *Cryptobytes (RSA Laboratories, Summer/Fall 2002)*, 5(2):2–13, 2002.

[24] D. Qu, B. M. Vetter, F. Wang, R. Narayan, S. F. Wu, Y. F. Jou, F. Gong, and C. Sargor. Statistical anomaly detection for link-state routing protocols. In *Proceedings of 1998 International Conference on Network Protocols*, Austin, TX, October 1998.

[25] J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[26] B. Schneier. *Secrets & Lies: Digital Security in a Networked World*. John Wiley & Sons, Inc., 2000.

[27] B. R. Smith, S. Murthy, and J.J. Garcia-Luna-Aceves. Securing distance-vector routing protocols. In *Proceedings of Internet Society Symposium on Network and Distributed System Security*, pages 85–92, San Diego, California, February 1997.

[28] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley. Leader election algorithms for wireless ad hoc networks. In *The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, April 2003.

[29] M. G. Zapata. Secure ad hoc on-demand distance vector (SAODV) routing. IETF Internet Draft, draft-guerrero-manet-saodv-00.txt, August 2001 (Work in Progress), August 2001.

[30] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '02)*, Lausanne, Switzerland, June 2002.

[31] L. Zhou and Z. J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, Nov/Dec 1999.

APPENDIX

A. SELECTED FEATURES

The features (used in classification) constructed in our experiments belong to two categories, non traffic-related and traffic-related. All non traffic-related features are detailed in Table 4 and the meaning of each feature is further explained in the "Notes" column. These features capture the basic view of network topology and route fabric update frequency. They are calculated based on the scenario and mobility scripts and the trace log file.

All traffic-related features are collected under the following considerations. Packets come from different layers and different sources. For example, it can be a TCP data packet delivered from the originator where the feature is collected. It can also be a route control message packet (for instance, a `ROUTE REQUEST` message, used in AODV and DSR), which is being forwarded at the observed node. We can then define the first two aspects of a traffic-feature as, *packet type*, which can be data specific and route specific (including different route messages used in AODV and DSR), and *flow direction*, which can take one of the following values, `received` (observed at destinations), `sent` (observed at sources), `forwarded` (observed at intermediate routers) or `dropped` (observed at routers where no route is available for the packet). We do, however, exclude the combination that data packets can be forwarded or dropped since it would never appear in a real ns-2 trace log. Routing protocols in MANET usually encapsulate data packets by adding particular headers with routing information at the source node and unpack them at the destination. Therefore all activities (including forwarding and dropping) during the transmission process only involve "route" packets. Also, we need to evaluate both short-term and long-term traffic patterns. In our experiments, we sample data in three predetermined *sampling periods*, 5 seconds, 1 minute and 15 minutes. Finally, for each traffic pattern, we choose two typical *statistics measures* widely used in literature, namely, the packet count and the standard deviation of inter-packet intervals. Overall, a traffic feature can be defined as a vector $<$ packet type, flow direction, sampling periods, statistics measures $>$. All dimensions and allowed values for each dimension are defined in Table 5. For instance, the feature to compute the standard deviation of inter-packet intervals of received `ROUTE REQUEST` packets every 5 seconds can be encoded as $< 2, 0, 0, 1 >$. Overall, we have $(6 \times 4 - 2) \times 3 \times 2 = 132$ traffic features, where 6, 4, 3, 2 are the number of packet types, flow directions, sampling periods and statistics measures, respectively.

For all continuous features or discrete features with infinite value space, we discretize them using a frequency-bucket scheme. We divide the value space of a continuous feature into a fixed number of continuous ranges (buckets), so that the frequencies of occurrences of feature values dropped in all buckets are equal. Then a continuous feature can be replaced by the index of its corresponding bucket. This approach guarantees that the chances of appearance of all possible labels (after discretization) in a feature are approximately the same. A pre-filtering process using a small random subset of normal vectors is necessary to retrieve the frequency distribution of all continuous features. In our experiments, we choose the bucket number to be 5.

Table 4: Topology and route related features

Features	Notes
time	ignored in classification. Only for reference
velocity	node movement velocity (scalar)
route add count	routes newly added via route discovery
route removal count	stale routes being removed
route find count	routes in cache with no need to re-discovery
route notice count	routes added via overhearing
route repair count	broken routes currently under repair
total route change	route change rate within the period
average route length	average length of active routes

Table 5: Traffic related features

Dimension	Values
Packet type	data, route (all), `ROUTE REQUEST`, `ROUTE REPLY`, `ROUTE ERROR` and `HELLO` messages
Flow direction	received, sent, forwarded and dropped
Sampling periods	5, 60 and 900 seconds
Statistics measures	count and standard deviation of inter-packet intervals