# Multigrade Security Monitoring for Ad-Hoc Wireless Networks

Matthew Tan Creti, Matthew Beaman*, Saurabh Bagchi, Zhiyuan Li*, Yung-Hsiang Lu
*School of Electrical and Computer Engineering, Department of Computer Science(*)*
*Purdue University*
*West Lafayette, IN 47907*
*Email: {mtancret,mbeaman,sbagchi,li,yunglu}@purdue.edu*

## Abstract

*Ad-hoc wireless networks are being deployed in critical applications that require protection against sophisticated adversaries. However, wireless routing protocols, such as the widely-used AODV, are often designed with the assumption that nodes are benign. Cryptographic extensions such as Secure AODV (SAODV) protect against some attacks but are still vulnerable to easily-performed attacks using colluding adversaries, such as the wormhole attack. In this paper, we make two contributions to securing routing protocols. First, we present a protocol called Route Verification (RV) that can detect and isolate malicious nodes involved in routing-based attacks with very high likelihood. However, RV is expensive in terms of energy consumption due to its radio communications. To remedy the high energy cost of RV, we make our second contribution. We propose a multigrade monitoring (MGM) approach. The MGM approach employs a previously developed lightweight local monitoring technique to detect any necessary condition for an attack to succeed. However, local monitoring suffers from false positives due to collisions on the wireless channel. When a necessary condition is detected, the heavy-weight RV protocol is triggered. We show through simulation that MGM applied to AODV generally requires little extra energy compared to baseline AODV, under the common case where there is no attack present. It is also more resource-efficient and powerful than SAODV in detecting attacks. Our work, for the first time, lays out the framework of multigrade monitoring, which we believe fundamentally addresses the tension between security and resource consumption in ad-hoc wireless networks.*

## 1. Introduction

Wireless networks depend on multi-hop routing to communicate beyond a single radio's range. Multi-hop routing relies on trusting nodes in the network to faithfully forward packets. Because it is possible that some nodes in the network are physically compromised by an adversary, methods for securing routing from misbehaving nodes is required. Cryptographic methods, such as Secure AODV (SAODV), provide some benefit in securing routing protocols [1]. For example, routing protocols often use a hop count to choose a least-cost route. Each time a packet is forwarded, the forwarding node must increment the hop count by one. SAODV employs a cryptographic method of using hash chains to prevent a malicious node from decreasing the hop count of a packet it forwards—for example, if the malicious node wants to attract routes go through it. However, SAODV cannot defend against all types of attacks. For example, it only works under the assumption that nodes that are further than one radio hop away from each other are not able to directly share information though an out-of-band channel, such as in the wormhole attack. The *wormhole attack* is where malicious nodes that are farther than a normal radio range away from each other use an out-of-band channel, such as a wireline connection, to tunnel packets between each other. Note that wormhole attacks can be launched by insider adversaries by changing routing packets and this is as yet an open problem how to efficiently detect and isolate such malicious nodes.

The complement to cryptographic security is intrusion detection. An example of intrusion detection is *local monitoring* [2]–[4]. In local monitoring, nodes overhear communication in the network and through this, observe the behavior of neighboring nodes. For example, local monitoring can be used to detect if a neighbor is delaying, dropping, modifying, fabricating, or misrouting packets. Local monitoring is a lightweight form of intrusion detection, because no extra radio communication is generated so long as no misbehavior in the network is detected. It can also be used to detect misbehavior in the presence of wormholes [2]. It has a very low false negative detection rate, however, it suffers from a high false positive detection rate due to the imperfection of wireless radio channels. It is also not able to detect sophisticated attacks with multiple colluding nodes, which may, for example, share the load of tunneling packets.

In this paper we present two innovations. First, we present a new protocol called *Route Verification (RV)* for detecting the wormhole attack against routing protocols. RV has no false positive and a low false negative rate. However, it is expensive in terms of energy consumption

978-1-4244-5113-5/09/$25.00 ©2009 IEEE

due to its radio communications. So we present our second innovation—a framework for monitoring called *Multigrade Monitoring (MGM)*. The specific instantiation of the framework we describe in this paper—also called MGM as a short-hand—uses the light-weight local monitoring to detect the necessary condition for the attack. The detection is used as a trigger for the heavier weight RV protocol.

We demonstrate our innovations through simulation, using TOSSIM [5] in TinyOS, of the AODV routing protocol. We compare MGM and RV to the baseline AODV and to SAODV. In general, MGM requires little extra energy compared to AODV during the benign phase and is more resource-efficient and powerful than SAODV in detecting attacks.

## 2. Background

### 2.1. Local Monitoring

Local monitoring uses a fundamental characteristic of wireless communications, that packets are easily overheard, to observe and detect misbehaving nodes. Misbehavior is broadly classified as any violation of the packet forwarding function, namely, delaying, dropping, modifying, fabricating, or misrouting packets. An early example of local monitoring in wireless ad hoc networks is the watchdog scheme [3]. This scheme applies to any protocol where a node is expected to forward a packet, after receiving it, within a set amount of time. When node $A$ sends a packet to node $B$, $A$ expects to overhear $B$ forwarding the packet to the next hop. If this does not happen $A$ suspects $B$ of misbehavior. A severe limitation of local monitoring is that wireless channels are not perfect and they suffer from losses and collisions. Suppose that $B$ does forward the packet, but a collision with another packet causes $A$ to fail to overhear the packet from $B$. For this reason, local monitoring must rely on a probabilistic approach to detect misbehavior such as seen in LITEWORP [2].

In LITEWORP, all nodes in the network participate in monitoring the behavior of their neighbors [2]. Whenever a node $G$, that is a neighbor to both nodes $A$ and $B$, overhears a packet sent from $A$ to $B$, $G$ becomes a *guard node* of $B$. The guard node $G$ expects to overhear node $B$ forward the packet, unmodified, to the next hop within a set amount of time. If this does not happen, $G$ suspects $B$ of malicious behavior. Another malicious behavior that can be detected by local monitoring is packet fabrication. If $B$ sends a packet as if it were forwarding from $A$, then node $G$ will overhear the packet from $B$ but not from $A$, and it will suspect node $B$ of fabricating the packet. In order to detect fabrications in this manner it is required that a node indicate where it is forwarding a packet from (the previous hop). If a malicious node purports forwarding from a node that is not an actual neighbor to itself, then there will be no guard node (i.e. a node that is a neighbor to both the malicious node and the purported previous hop) that is able to suspect the malicious node of packet fabrication. For this reason, it is assumed that a secure neighbor discovery protocol is used by all nodes, for example [6], to determine their neighbors and their neighbor's neighbors.

In LITEWORP, a packet forward event is monitored by multiple guard nodes; the number depends on the local node density. When the number of suspicious events committed by one node within a window of time exceeds a threshold, then the node is determined to be malicious. A quorum of the guards need to agree that a node is malicious in order to isolate it from the network. Both these steps are meant to distinguish between natural failures due to the imperfect wireless channel and malicious actions. Nevertheless, LITEWORP is still susceptible to false detection due to the imperfect wireless channel, and particularly so when collisions are frequent.

### 2.2. Wormhole Attack and Defenses

One use of local monitoring is to detect a particularly severe attack known as the wormhole attack. In the *wormhole attack* an adversary tunnels packets between parts of the network that are not normally one hop away from each other (given the radio capabilities of the benign nodes in the network). This could be used to insert a malicious node into a route. For example, suppose AODV is being used, and an adversary has a malicious node $M_1$ near the origin node that is trying to establish a route and another malicious node $M_2$ near the route destination node. When $M_1$ overhears the origin sending a route request it tunnels the packet to $M_2$, say through an out-of-band channel. Node $M_2$ forwards the route request as if it had received it from a neighbor but with a lower hop count than if it had received the packet through a legitimate route. When the destination receives the packet with low hop count it will choose the reverse route though $M_2$ to send the route reply. $M_2$ may drop the route reply causing route establishment to fail or it may forward the route reply though the tunnel to $M_1$, allowing a route to form over which it has control—it may then selectively drop any data packet passing along the route.

Cryptographic methods are not enough to defeat the wormhole attack. For example, secure AODV (SAODV) is nearly as defenseless against the wormhole attack as baseline AODV. In SAODV an origin node signs the non-mutable parts of a route request packet [1]. Each forwarding node verifies the packet before modifying its route tables and forwarding the packet. The hop count is protected by attaching a value called the $hash$ and a value called

$top\_hash$ to a route request. The origin node computes $top\_hash = h^{MAX\_HOP\_COUNT}(hash)$, where $hash$ is a random value selected by the origin node, and using a secure hash function $h$. Both $hash$ and $top\_hash$ are attached to a route message. Each time a node forwards a packet it changes the value of $hash$ to be $hash = h(hash)$. When a node receive a packet it authenticates the hop count by verifying that $top\_hash = h^{MAX\_HOP\_COUNT-hop\_count}(hash)$. Because a node knows only the value of $hash$ from the node it receives the message from, and not previous hops, the node can increment the hop count by zero or more before forwarding, but it cannot decrement hop count. However, this assumes there is no colluding adversaries that are able share information using an out-of-band channel, as in a wormhole attack. Colluding adversaries can pass the packet through the tunnel without incrementing the hop count and without the possibility of detection through SAODV.

We will define packet *fabrication* as the following event—a node gives the appearance of forwarding a packet when that packet was not actually sent by any of the node's neighbors. By this definition the wormhole attack, where packets are tunneled between distant parts of the network, *always involves a packet fabrication*. We are not interested in defending against physical layer wormholes, where the adversary does not have to change any packet, since any secure neighbor discovery such as [6] can eliminate physical layer wormhole attacks. RV and MGM defend against routing layer wormhole attacks, where the adversary must modify the contents of a routing packet when replaying it at the wormhole end-points.

## 3. Multigrade Monitoring Framework

For monitors[1] having high false positive rates, it is often necessary to set thresholds, such that detection occurs only if the rate of suspicious events exceeds some threshold. If this were not done, then a system that isolates detected malicious nodes, such as LITEWORP, would likely isolate most of the nodes in the network. Instead consider the scenario with two monitors $M_1$ and $M_3$, where $M_1$ has a high false positive rate, but is low cost, and is able to detect the necessary condition for an attack. By necessary condition, we mean that this step needs to be achieved for the attack to succeed. Monitor $M_3$ is more accurate than $M_1$, i.e., has a much lower false positive rate, but is also more expensive resource usage-wise. In the Multigrade Monitoring (MGM) framework, multiple monitors such as $M_1$ and $M_3$ are used together, by having $M_1$'s detections trigger $M_3$'s actions.

More formally, we notate monitor $i$ as $M_i$, with false positive rate $FP(M_i)$, false negative rate $FN(M_i)$, and

1. We will use the term "monitor" to mean a detection technique.

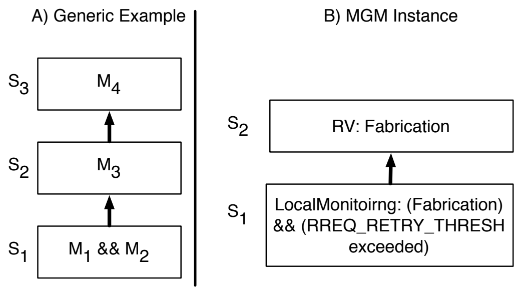

Figure 1. (a) Example of multigrade monitoring, where stage $S_1$ triggers $S_2$ which in turn triggers $S_3$. Stages are ordered such that $FP(S_1) > FP(S_2) > FP(S_3)$. (b) Specific instantiation of this framework that we describe in Section 4.

the resource cost to monitor a single event as $C(i)$. Assume, without loss of generality and for a clean exposition, that the cost for a monitor to monitor an event is the same for all events. As shown in Figure 1, monitors are grouped into stages $S_1$ to $S_n$, where a lower numbered stage triggers a higher numbered stage when the lower stage detects a triggering event. When the final stage $S_n$ detects a malicious event, diagnosis and attack mitigation is triggered. In this paper, diagnosis and attack mitigation means discovering the malicious node(s) and isolating them from the network. A stage may contain multiple monitors grouped together in such a way that for the stage to trigger the next stage, all the monitors must trigger at the same event. We make the following observations.

1) As long as every stage has zero false negatives (i.e., missed alarms) then the monitoring system will have no false negative.
2) If the stages are ordered by false positive rate such that, $FP(S_1) > ... > FP(S_n)$, the the false positive rate of the system will be no greater than $FP(S_n)$.
3) Combining points 1 and 2, if the false negative rate at all stages is zero and the false positive rate of $FP(S_n)$ is also zero, then the system has perfect detection.
4) Define the rate of monitored events as $r$. The cost to the system, in the benign case where no attack is present, is $r*C(S_1)+r*FP(S_1)*C(S_2)+...+r*FP(S_1)*...*FP(S_n)*C(S_n) << r*C(S_n)$.

In the next section we present a system that fits into the MGM framework. Local monitoring is used as a low cost monitor with high false positives but low false negatives. A new protocol called Route Verification (RV) provides low false positives and false negatives. These protocols are combined in the MGM framework. We believe that many existing techniques for security in wireless networks can be mapped to the MGM framework. As another example, Sybil detection [7] is meant to detect if multiple identities are being used in a wireless network. This can be done when there is a collision of claimed IDs from multiple nodes. MGM represents a way to structure security in wireless networks, which balances the competing goals of detecting a large class of attacks and do the detection using low resource usage.

## 4. Route Verification (RV) Protocol

### 4.1. Attack Model

We consider the adversary has the goal of disrupting route establishment, i.e., an origin node is not able to establish a route with a destination node. The underlying model is that the origin tries route establishment a certain number of times (denoted `MAX_ROUTE_RETRIES`) before giving up. So the attack model is the route request has to be disrupted these many times. Arbitrary collusion among the malicious nodes is possible.

To achieve this, the following three basic actions can be adopted by the adversary: (i) modification of non-mutable parts of the packet; (ii) fabrication of a packet; (iii) not correctly incrementing the hop count on forwarding a packet. All other actions to disrupt AODV route establishment have been shown in [8] to be mappable to these. We only consider attacks that change the contents of routing packets and not timing-based attacks or Denial of Service attacks.

### 4.2. Assumptions

First, we assume that a secure neighbor discovery protocol is being used, so that nodes know their neighbors and their neighbor's neighbors. An example of secure neighbor discovery is [6]. We define *neighbors* as pairs of nodes that are able to directly receive each others packets using radios that are common to the set of legitimately deployed nodes. Note that neighbor discovery, while a necessary step, does not solve the problem of attacks that we target in this paper. As a simple example, two ends of the wormhole can fabricate packets to insert themselves in a route despite neighbor knowledge at all legitimate nodes.

Second, we assume that the network is not highly mobile. The Route Verification (RV) protocol relies on tracing the path back to the source of a packet. If the network has a high degree of mobility, enough nodes along the path between the packet's origin and the node initiating trace may have moved between the time that the originator generates the request and it receives the response, such that, tracing the reverse route of the packet is no longer possible. If this were the case then AODV would likely fail to establish a route anyway, because AODV requires the route reply packet to follow

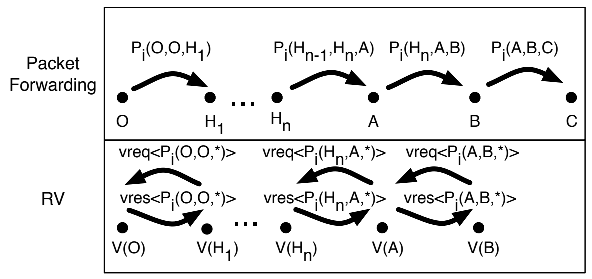

Figure 2. Example of RV used to verify packet $P_1$ at each hop using verify request (vreq) and verify response (vres) messages. The chain of *vreq*'s is started by node $V(B)$.

Table 1. The verify table stores an entry for each packet instance.

| Verify Table | | |
|---|---|---|
| Field | Type | Description |
| heard_from | address | The node the packet is overheard from. |
| addressed_to | address | Where the packet was addressed. |
| non_mutable_hash | hash | A hash of the non-mutable fields (e.g. origin and destination addresses) of the packet. |
| mutable | list | The mutable fields (e.g. hop count) of the packet. |
| state | enumeration | Can be {`unverified`, `verified`, `invalid`}. |
| verify_pending | boolean | True when a verification request has been sent. |
| expires | timestamp | Time when the entry can be removed from the table. |

a reverse path from the destination node to the origin to establish a route.

Third, we assume that the packets that are to be protected by our protocols are authenticated hop-by-hop. That is, when $X$ forwards a packet, it attaches a key to the packet that only it could know, and that all of $X$'s neighbors can use it to authenticate that $X$ is the node forwarding the packet. This type of authentication can be done using one way hash chains as presented in [9]. This assumption is necessary in local monitoring to prevent a malicious node from lying about its identity, when committing a malicious action, in order to frame an innocent node. Also, for non-repudiation, when a node $Y$ forwards a message from $X$ to $Z$, node $Y$ attaches the key that $X$ used to authenticate itself. This means $Y$ cannot be accused of fabrication.

### 4.3. Route Verification Overview

The goal of Route Verification (RV), is to ensure that any fabrication or incorrect modification of a forwarded packet will be detected; and that the neighbors of the node committing the incorrect action will correctly diagnose that node as the source of the problem. RV achieves this by using a combination of passive local monitoring to observe packets being forwarded, and active requests to previous hops in a path to validate a packet. We use the term *packet instance* to define the contents of a packet when it is forwarded from a particular node. The notation for a packet instance is $P_i(X, Y, Z)$ where $P_i$ is packet $i$, $X$ is the node from which node $Y$ received packet $P_i$, $Y$ is the node forwarding $P_i$, and $Z$ is the node to which $Y$ is forwarding $P_i$. In the case of a broadcast packet, (e.g. route request in AODV) the packet instance is written $P_i(X, Y, *)$ where * means any node. Any packet has mutable fields and non-mutable fields. Mutable fields have values that change at each hop, such as the hop count; non-mutable fields have values that do not.

The *verify table* for each node, shown in Table 1, contains an entry for each packet instance overheard by that node. A verify table entry for a packet instance can take on one of three possible states `unverified`, `verified`, or `invalid`. The state `unverified` is the default state for a new packet instance, `verified` means that the packet instance and all previous instances of the packet back to an origin node $O$ have been correctly forwarded, and `invalid` means that it is known that the packet instance or a previous packet instance has been fabricated or modified.

After some node $B$ receives the packet $P_i$ from node $A$ and forwards it to node $C$, all the neighbors of $B$ create an entry for packet instance $P_i(A, B, C)$ in their verify tables and set the state to the default, `unverified`. One of the neighbors of $B$ is designated the *verifier node*, we will denote it as $V(B)$. Choosing the verifier node is explained in Section 4.4.1, and the case where the verifier may itself be malicious in Section 4.4.2. Verifier $V(B)$ multicasts a verification request. A *verify request* message contains a record of the mutable fields and a hash of the non-mutable fields of the packet instance that is being verified and is notated as $vreq < packet\_instance >$.

We use a reliable authenticated multicast functionality as described in [10]. The multicast is limited to nodes that are a particular number of hops from another node. The multicast is notated as $mcast(< message >, n, h)$, where $h$ is the number of hops from the node $n$ that the message will reach. The verify request is multicast as $mcast(vreq < P_i(A, B, *) >, B, 2)$; all 1 and 2 hop neighbors of $B$ receive the message. This ensures that all the neighbors of node $B$ and of node $A$ receive the message. It is critical that all neighbors of $B$ receive the message so that they know $V(B)$ has correctly fulfilled its duty of sending a verify request; and all neighbors of $A$ must receive the message so that they know a request has been received for which a response is expected.

Upon receiving the verify request message, the neighbors of $B$ set `verify_pending` to true for that packet instance in their verify tables. One of the neighbors of $A$ is designated the verifier for $A$ called $V(A)$, checks its verify table for any packet instance $P_i(*, A, B)$. A matching packet instance would have the same `non_mutable_hash` as the packet instance in the request and would have been forwarded from $A$ to $B$. If $V(A)$ finds a match, and the state of the matching entry is set to `verified`, it multicasts a verify response to the neighbors of nodes $A$ and $B$; $mcast(vres < P_i(A, B, *), verified >, A, 2)$. A verify response message contains the packet instance for which it is responding and a diagnosis from the set {`verified, invalid, fabricated, modified`}. It is notated as $vres < packet\_instance, diagnosis >$. If $V(A)$ finds a match with the state set to `invalid`, then V(A) sends a multicast $mcast(vres < P_i(A, B, *), invalid >, A, 2)$. Diagnosis `Invalid` indicates that the packet has been modified or fabricated by a previous hop. If $V(A)$ finds a match with the state set to `unverified` and the previous hop is $H_n$, then the verifier creates a verification request by performing $mcast(vreq < P_i(H_n, A, *) >, A, 2)$. When $V(A)$ receives a verify response for $P_i(H_n, A, *)$ it will then send the appropriate verify response, either `verified` or `invalid`, for $P_i(A, B, *)$.

Figure 2 shows how packet $P_i$ is verified by the neighbors of each hop. When node $O$ originates packet $P_i$, all of the neighbors of $O$ overhear the packet instance $P_i(O, O, H_1)$ and record it in the verify table with the state set to `verified`. When $H_1$ forwards the packet, the verifier $V(H_1)$ sends a verify request and the verifier $V(O)$ sees that the state has been set to `verified` and sends a verify response with the diagnosis `verified`. From this we can state the following two invariants:

***Inv1:*** If any node has set a packet instance to `verified`, then all nodes that have records of previous instances of the same packet (i.e., on previous hops) have also set the state of those packet instances to `verified`.

***Inv1:*** The only way the neighbors of node $O$ set packet $P_i(O, O, H_1)$ to `verified` is if they have overheard the message directly from node $O$.

We can put together these two invariants to state the following lemma:

***Lemma:*** If a packet instance at hop $i$ is marked `verified`, then that the packet has not been fabricated or modified till the hop $i$.

*Attack Scenario*

The diagnosis is performed by the verifier node that sends the verify response message. For example, consider the case where $V(A)$ has received a request for packet instance $P_i(A, B, *)$. It does not have a match for $P_1(*, A, B)$ in the verify table, and it initiates a verify request $vreq < P_1(*, A, B) >$, which is not responded to before a request timeout period. This will detect a fabrication. $V(A)$ multicasts a response $mcast(vres < P_1(A, B, *), fabricated >, A, 2)$. All the neighbors of $B$ receive this response and isolate node $B$ from the network.

## 4.4. Route Verification Details

**4.4.1. How is the verifier node chosen?** The verifier node of a node has a central role in initiating the verification requests and then relaying verification responses. It is possible that the verifier node fails to overhear a packet forwarded from say $X$, which then causes it to fail to send a verify request. For this reason, all neighbors of a node $X$ can take the role of verifier if they detect that the "head verifier" has failed to send a verify request or verify response. The *head verifier* of $X$ is notated as $V_1(X)$ and the other neighbors are given an order in the verifier hierarchy $V_2(X)$ to $V_n(X)$. If a verify request or response should be sent and verifier $V_1(X)$ fails to send the request or response within a given response time then $V_2(X)$ will send the request or response and so on until $V_n(X)$'s time slot is reached.

The order of verifiers is created using a secure hash function $H$. For each neighbor $i$ of $X$, take the hash of $i$ concatenated to $X$. This value, H($i$,$X$), is the *verifier id* of $i$ for node $X$. Assign $V_1(X)$ as the node with the smallest verifier id, $V_2(X)$ as the node with the next smallest verifier id, and so on. In the unlikely case of a tie, the node with the smaller node id wins. This method for assigning verifier ids helps to distribute the role of verifier evenly among the nodes, and makes it difficult for a malicious node to make itself the verifier node for a large number of other nodes.

**4.4.2. What happens if a verifier node is malicious?** If a verifier is malicious and does not send a required request or response, this is handled as we saw in Section 4.4.1. If a node suspects another node of sending a malicious verify response, it multicasts a challenge for that response. A malicious verify response may be responding that a packet instance has been verified, when in reality it has not. Note that all nodes in a neighborhood agree on whether or not a packet instance has been verified based on Inv1. Suppose that for node $A$, $V(A)$ sends out a malicious response. Another neighbor of $A$, say $V_2(A)$ makes the challenge and a vote happens among all neighbors of $A$. An example voting scheme is shown in [11]. If a quorum of the votes say the response message is malicious then the node that sent the message is isolated from the network, otherwise the node that requested the vote is isolated from the network. Conversely for a malicious request, if a node $B$ detects a malicious request for a message that it forwarded, it is $B$'s responsibility to initiate the challenge. This means that for RV to work, no node can have a quorum or more number of malicious neighbors. Voting is costly because it requires every neighbor to multicast an authenticated ballot; but an adversary would not be able to take advantage of the voting process to consume resources for long, because the outcome of the vote is always one malicious node being isolated from the network.

**4.4.3. What happens when the previous hop is unknown?** Suppose that the verifier for node $A$ receives the verify request $vreq < P_i(A, B, *) >$ and it has no entry in the verify table matching $P_i(*, A, B)$. This could mean that the packet was fabricated or that the verifier node failed to overhear $A$ forwarding the packet to $B$ due to a natural failure. The verifier will send a multicast $mcast(vreq < P_i(*, A, *) >, A, 2)$. All nodes receiving the broadcast check for a match to $P_1(*, *, A)$ in their verify table. If a match is found they send a verify response. If a match is not found, then after a timeout period, the verifier for node $A$ assumes the message was fabricated and sends the verify response indicating that $B$ fabricated the packet. The chance that *all nodes* failed to overhear the message and yet the transmission from $A$ to $B$ happened successfully is small.

**4.4.4. Can a malicious node use radio control to frame a legitimate node of fabrication?** Suppose that malicious node $M$ controls the power of its radio so that only node $B$ overhears a packet it fabricates. When $B$ forwards the packet to say $C$, the verifier of $B$ will generate a verify request $vreq < P_i(M, B, *) >$. The verifier of $M$ will see that $M$ neither received nor forwarded any packet $P_i$ and will send the verify response $vres < P_i(M, B, *), fabricated >$. This will cause $B$ to be incorrectly isolated from the network. To provide non-repudiation of a forwarded message, a node attaches the key that was used by the previous hop. To prevent this from happening, when the verifier of $B$ sends the verify request $vreq < P_i(M, B, *) >$, it includes the key that $M$ used to authenticate itself (recall the non-repudiation design mentioned in Section 4.2). When the neighbors of $M$ receive the verify request with the key attached, they will be able to verify that only $M$ could have known this key. From this they can correctly diagnose that if packet instance $vreq < P_i(M, B, *) >$ is fabricated, then it is node $M$ that performed the fabrication, and not node $B$ which innocently forwarded the packet received from $M$.

#### 4.4.5. What type of routing and forwarding protocols can RV secure?

As described here RV can protect route request, route response, and data messages in any distance vector routing protocol, such as AODV. The difference between route request and response is that the request is a broadcast message. So when node $B$ forwards a route request it would look like $P_1(A, B, *)$ rather than $P_1(A, B, C)$. This does not change the working of RV in any way. RV can also secure a beacon routing protocol like those commonly found in sensor networks. Consider beacon routing where each node periodically announces its minimum hop count distance from a base station node. If we add the rule that a node must announce the neighbor with the lowest hop count along with its beacon message, then RV can generate a verify request message where the previous hop is taken to be the neighbor with the lowest hop count. With slight modification, RV can also secure aggregation protocols commonly used in sensor networks. For aggregation, rather then sending the verify request for a single previous hop, the verify request would be sent for every node that contributed to the aggregated message.

RV can generate a very large number of verify request messages when securing a broadcast packet such as a route request in AODV. For example, consider the case where node $D$ broadcast a route request packet that is then forwarded by three neighbors: $E$, $F$, and $G$. This would cause verifier nodes $V(E)$, $V(F)$, and $V(G)$ to generate route requests $vreq < P_j(D, E, *) >$, $vreq < P_j(D, F, *) >$, and $vreq < P_j(D, G, *) >$ respectively. All three requests would be responded to by the same diagnosis from verifier $V(D)$. It would be more efficient if just one of the verifiers sent a route request. This is done by waiting a random backoff time before sending verify request and suppressing the request if a response is already heard.

## 4.5. Incorporating RV in MGM

Although RV is able to correctly detect and diagnose all packet fabrications and modifications, it is very costly in terms of radio traffic generated. On the other hand, LITEWORP which is based on passive local monitoring generates no extra traffic except when malicious activity is detected. However it has two sources of weakness as discussed in the introduction—it requires a malicious node to generate anomalous activity above a particular rate for detection and colluding malicious nodes can stay below the threshold for each individual node by sharing the activities; it has false positives due to link losses and collisions. Multigrade Monitoring (MGM) combines LITEWORP and RV to obtain a protocol that can correctly detect and diagnose all route establishment attacks lasting longer than a threshold amount of time, and does so with nearly the same low cost of local monitoring protocols. To see what this threshold amount of time is, consider the following scenario.

Assume that AODV has been implemented to retry a route request every `RREQ_RETRY_RATE` seconds until a route reply is received indicating route establishment. Observing repeated route request `RREQ_RETRY_RATE` seconds apart for a given source-destination pair is an indication that an adversary is maliciously preventing route establishment. The same is true for LITEWORP suspecting malicious packet fabrications. While neither repeated route requests nor suspected fabrications is a *sufficient* condition for concluding that a wormhole attack is occurring, each is a *necessary* condition for the attack.

We set a threshold `RREQ_RETRY_THRESH`, such that if a node overhears that many route requests for the same source-destination pair within `RREQ_RETRY_THRESH*RREQ_RETRY_RATE` and the node suspects even a single packet fabrication using local monitoring, the node will enter into RV mode and initiate the verify requests. Thus, RV will detect route disruption attacks that last longer than `RREQ_RETRY_THRESH*RREQ_RETRY_RATE`.

## 4.6. Coverage of Attacks in MGM

Recall from section 4.1, that our goal is to prevent an adversary from disrupting route establishment. We want to detect and isolate malicious nodes that are disrupting the route before the origin has performed `MAX_ROUTE_RETRIES` route requests. For this, MGM needs to achieve two properties—first, MGM should trigger before `MAX_ROUTE_RETRIES` and second, RV should detect the route disruption attack and specifically, any of the three adversarial actions for route disruption that we introduced in Section 4.1.

For the first property, MGM sets `RREQ_RETRY_THRESH` to be less than `MAX_ROUTE_RETRIES`. For the second property, local monitoring can detect modification of the non-mutable parts of a packet or an incorrect increment (including no increment) to a hop count with vanishingly low false alarm. Therefore, RV need only concern itself with fabrication. As described in the RV design, a suspected fabrication results in verification requests propagating all the way back to the origin node. The neighbors of the origin only verify a packet if they have directly heard it. As long as only less than a quorum of nodes in any neighborhood along the path from the origin to the destination is malicious, then the verification responses are received correctly and any fabricated packet is detected. There is a fairly obvious implication to this quorum requirement. If the quorum threshold is set too low, then an adversary can compromise greater than that threshold number of nodes in a neighborhood. If it is set too high, then there may not be enough density in the network for RV to make progress. This is a design parameter that needs to be set by the network operator based on her deployment. Regarding the applicability of MGM to a routing protocol, if in the routing protocol a packet is modified by a node in a manner that cannot be deduced by its neighbors, then MGM will not work. An example of such a protocol is the onion anonymous routing.

## 5. Experiments and Results

Our results are based on simulations in the TinyOS 2.0.2 version of TOSSIM. We simulate 200 nodes, and vary the size of a square region to obtain topologies with different average degrees (number of neighbors) per node. The version of TOSSIM models packet loss of the CC2420 radio based on signal to noise ratio (SNR), where the noise is generated from real noise traces using a technique called closest-fit pattern matching [12]. Thus inputs to this radio model are a noise trace (we use the Meyer Library trace) and the link gain, in both directions, between every pair of nodes in the network. Link gains between every pair of nodes is based on the distance between nodes and is generated using the TinyOS support code `LinkLayerModel.java`. To generate random topologies, we randomly distribute nodes in a square region and then generate their link gains. We then set any pair of nodes with a link gain greater than -70dBm in both directions to be neighbors. A node processes messages from its neighbors and discards others. The threshold of -70dBm provides reliable links between neighbors, given that the default value of the Clear Channel Assessment (CCA) threshold is -77dBm for CC2420 [13]. In reality, a node will not know the gains and will use some form of neighbor discovery to obtain the neighbor relations.

When the CC2420 radio is in receive mode it consumes about 35.5mW, which is even greater than the 31mW it consumes when it is transmitting at 0dBm [13]. For this reason, some form of low-power listening (where a radio is able to sleep between packet receptions) is required for nodes to conserve energy. We implemented the BMAC [14] protocol with low-power listening and added it to the TinyOS 2.0.2 version of TOSSIM, which was previously lacking a low-power listening mode. In BMAC, radios are set to wake up at a check interval and perform clear channel assessment. If the channel is clear the radio goes back to sleep, otherwise if the channel is not clear, the radio remains awake until it has received a message or a timeout period is reached. To ensure that all nodes receive a packet, a transmitting node must transmit a preamble that is the length of one check period before transmitting the actual packet. Based on this, a radio may be in one of four states: sleeping, waking up, receive mode, or transmit mode. We keep a count of the total time a radio is in each of these states and use this to compute the total energy consumed by the radio. In all simulations we set the check interval to be 25ms and duty cycle to be 10%. TOSSIM does not simulate time for code execution. Therefore, there is no easy way to account for the energy of code execution, which in any case is insignificant (for route establishment) compared to that used in network traffic. We do count the energy expended in instructions related to message transmission and receipt. This is also very small relative to that for network traffic, in all cases except SAODV.

**Behavior of SAODV:** In practice SAODV takes about 3 seconds to authenticate a route message at each hop on a TelosB mote. We use TinyECC to provide fast elliptic curve authentication [15]. If SAODV were being used on real motes it would not even be able to form a 5 hop route in the route retry period of 20 seconds used in all of our simulations. Nevertheless, it is interesting to compare energy consumption of SAODV to the other protocols, ignoring the effect of added latency to route request and reply.

We looked at other candidates (apart from SAODV) for comparison in the simulation experiments. However, no satisfactory candidate came up, since existing protocols do not handle the gamut of attacks that MGM does or makes unrealistic assumptions of the wireless network. An example of the first is identifying wormholes by considering the graph topology and that only handles physical layer wormholes, but not the routing layer wormholes. An example of the latter is temporal leashes for identifying wormholes and that assumes extremely tight time synchronization (order of $\mu$s) which is costly to achieve in wireless networks.

**Behavior of AODV:** Unless explicitly stated otherwise, the following parameters are used in all the simulations. The test application picks a new source destination pair 5 hops away from each other every 40 seconds and attempts to establish a route. After an origin node has made a route request it expects a route reply within 20 seconds. If this does not happen the origin will retry the route request, up to 5 times. There is no data traffic in the simulation because it is not relevant to the solution. We also use `RREQ_RETRY_THRESH` value of 3.

### 5.1. Energy Consumption in Benign Network

We examine how well the different protocols scale with average node degree and number of hops between the origin and destination, when no malicious node is present. For comparison, consider that idle receiving power is 35.5 mW and the network is operating at 10% duty cycle. Instead of plotting energy, we plot the power since the former will depend on the arbitrary setting of simulation time. In our application, power

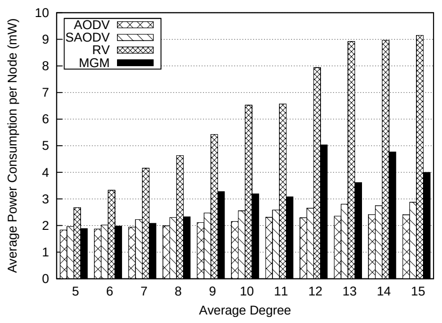

Figure 3. Average power consumption for different topologies.

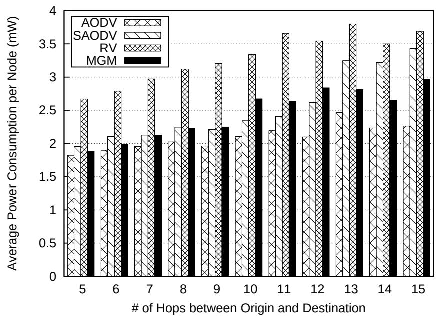

Figure 4. Average power consumption for different number of hops between origin and destination node in benign network.

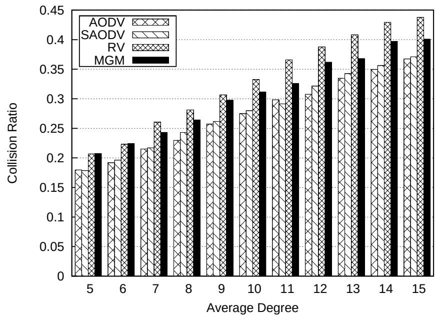

Figure 5. Collision ratio with average node degree for the different protocols. Collision ratio is the number of packet collisions to the number of packet collisions plus successfully received packets.

is directly proportional to energy. We see in Figure 3 that RV consumes power at a quickly increasing rate with increase in average node degree. This can be explained by the large number of verification requests and responses that are multicast up to 3 hops. For degrees less that 9, MGM consumes barely more power than the baseline AODV. Ideally, MGM will not trigger RV and will act the same as baseline AODV. As the degree increases, the probability increases that route requests will fail due to natural collisions, and hence LITEWORP will falsely suspect fabrications and MGM

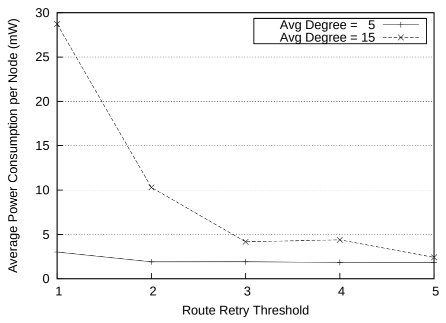

Figure 6. Average power consumption of the MGM protocol for different values of the trigger for invoking RV from MGM (`RREQ_RETRY_THRESH`). Two random topologies with degrees 5 and 15 are used.

will trigger RV. The ratio of packet collisions with degree in Figure 5 explains this increase. Both AODV and SAODV increase by only a small amount with increases in degree. In Figure 4 increasing hop count has a smaller affect on RV. SAODV's power consumption increases more rapidly with increasing hops than with increasing degree. This may be caused by the much larger packet size of SAODV which makes it more susceptible to losses on the longer paths. TOSSIM's packet loss model is such that a larger packet has a higher likelihood of overlapping with another packet, which raises the noise floor and ultimately leads to a higher likelihood of the packet being not received correctly.

## 5.2. Choosing MGM Parameters

A significant parameter in MGM is `RREQ_RETRY_THRESH`. This is the threshold number of times a node must observe an RREQ for a particular route establishment flow before triggering RV. Figure 6 shows how this threshold affects the average power consumption at a node. Setting the threshold to one means that every observed RREQ will trigger RV. This is very costly particularly for dense networks (node degree 15). However, for both sparse and dense networks, the cost decreases quickly and then stabilizes. This indicates that the performance of MGM is not very sensitive to this parameter setting in a reasonable range.

## 5.3. Isolation Coverage

In Figure 7 we see how increasing the amount of traffic in the network affects the fraction of malicious nodes that are detected and isolated. In this simulation the period between origin destination pairs attempting to establish a new route is varied. The probability of attacking an attempt to establish a route by a pair of malicious nodes forming a wormhole is $P$, which is set to 0.1 and 0.5. This mimics an intelligent adversary

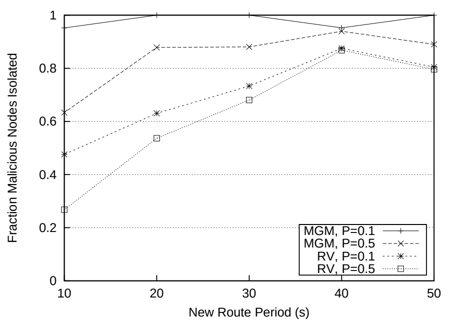

Figure 7. Isolation coverage for different amounts of load in the network.

which picks a subset of routes to attack to stay under the radar. The number of malicious nodes is not predetermined in these simulations, rather it is assumed that as many malicious nodes as are required to attack the corresponding number of route requests are available to the adversary. MGM performs better than RV since it uses the bandwidth more intelligently. The lack of coverage is caused by the route verification taking too long to complete. Expectedly the protocols perform better with fewer number of malicious nodes ($P = 0.1$ than 0.5).

## 6. Related Work

Sy and Bao present a packet traceback mechanism called CAPTRA that shares similarities with RV [16]. In CAPTRA all nodes in the network add overheard messages being forwarded by neighbors in a bloom filter. When an access point (e.g. a base station node or a destination node in AODV) detects a security breach it sends a traceback request message. Nodes use their bloom filter to look up the source of the packet for which the request has been made. If a node finds an entry in the bloom filter it sends an announcement of the source of the packet (the previous hop where the packet came from). When a quorum of nodes have announced the source of the packet, a trace request is sent to that source and the process repeats itself until the ultimate origin of the packet has been found. Trace confirmation messages are propagated back to the requesting access point at each hop of the trace, allowing the access point to diagnose the source of the security breach.

CAPTRA uses an *access point initiated trace* to find the source of an attack while RV uses a *locally initiated verification* to verify the source of a packet. The access point initiated approach would have less overhead in the case where a destination node is trying to find the source of a RREQ fabrication. However, locally initiated verification has the advantage of being able to isolate malicious nodes at the local level.

In [4] a locally initiated verification approach is used, but it is assumed that special secure monitoring nodes are available. RV can tolerate any nodes in the network becoming compromised, so long as no more that a quorum of nodes is compromised in any neighborhood.

Other papers have used the idea of setting triggers that initiate increased levels of monitoring. For example, in [17] when an anomaly is detected in application data the system begins collecting additional state information on the network. In [18] only a few nodes monitor for anomalous behavior until an anomaly is found, then all of the two-hop neighbors of the node that detected the anomaly begin monitoring. To our knowledge, no one has presented a multigrade monitoring type of framework, where low cost local monitoring is able to detect all necessary conditions for an attack to succeed, and then trigger more costly detection mechanisms that have low rates of false positive detection.

ODSBR has been proposed as a Byzantine resilient on-demand routing protocol [19]. Byzantine behavior includes wormhole attacks and collusion. In ODSBR both the route request and route reply are flooded throughout the network. This ensures, that as long as a fault free path exists between the source and destination, route establishment cannot be suppressed by faulty nodes dropping packets. A metric that factors in failures and adversarial behavior is used to select the least cost path to a destination node during the route reply phase. The metric is a weight that represents the reliability of each link along the path. A path is accumulated hop-by-hop during the route reply phase, so that when the source node receives the route reply, it will know the nodes and the weight of each link on that path. This requires each node to sign the accumulated route reply message. In addition each node is required to authenticate all of the signatures of a route reply message, so that a fabricated route reply with a low cost is not selected over a legitimate route reply. ODSBR is a routing protocol, with its own route selection metric. This is different from the route protection mechanisms we have presented in this paper, which can be added to existing routing protocols such as AODV.

## 7. Conclusion

We have shown a multigrade monitoring approach that has little energy overhead compared to baseline insecure AODV, while having a high rate of detection and isolation. Our approach fundamentally balances the conflicting pulls of achieving high security with low resource consumption. We have shown this for routing protocols through two protocols—a previous protocol LITEWORP and a new protocol RV incorporated into a multigrade monitoring (MGM) framework. In future work, we are applying the MGM framework to other security attacks. We are also developing compiler-based tools to populate a MGM framework, given implementations of the individual protocols.

## References

[1] M. G. Zapata, “Secure ad hoc on-demand distance vector (saodv) routing,” September 2006, http://personals.ac.upc.edu/guerrero/papers/draft-guerrero-manet-saodv-06.txt (Retrieved: Apr 5, 2009).

[2] I. Khalil, S. Bagchi, and N. B. Shroff, “Liteworp: Detection and isolation of the wormhole attack in static multihop wireless networks,” *Comput. Netw.*, vol. 51, no. 13, pp. 3750–3772, 2007.

[3] S. Marti, T. J. Giuli, K. Lai, and M. Baker, “Mitigating routing misbehavior in mobile ad hoc networks,” in *Proc. MobiCom ’00*, 2000, pp. 255–265.

[4] C. Tseng, P. Balasubramanyam, C. Ko, R. Limprasittiporn, J. Rowe, and K. N. Levitt, “A specification-based intrusion detection system for aodv,” in *Proc. Workshop on Security of Ad Hoc and Sensor Networks (SASN)*, 2003, pp. 125–134.

[5] P. Levis, N. Lee, M. Welsh, and D. Culler, “Tossim: accurate and scalable simulation of entire tinyos applications,” in *Proc. SenSys ’03*, 2003, pp. 126–137.

[6] R. Maheshwari, J. Gao, and S. Das, “Detecting wormhole attacks in wireless networks using connectivity information,” *Proc. IEEE INFOCOM ’07*, pp. 107–115, May 2007.

[7] J. Newsome, E. Shi, D. Song, and A. Perrig, “The sybil attack in sensor networks: analysis & defenses,” in *Proceedings IPSN ’04*. ACM New York, NY, USA, 2004, pp. 259–268.

[8] P. Ning and K. Sun, “How to misuse aodv: A case study of insider attacks against mobile ad-hoc routing protocols,” in *Ad Hoc Networks*, 2005, pp. 795–819.

[9] S. Zhu, S. Setia, and S. Jajodia, “Leap: efficient security mechanisms for large-scale distributed sensor networks,” in *Proc. CCS ’03*. New York, NY, USA: ACM, 2003, pp. 62–72.

[10] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler, “SPINS: Security protocols for sensor networks,” *Wireless networks*, vol. 8, no. 5, pp. 521–534, 2002.

[11] I. Krontiris, Z. Benenson, T. Giannetsos, F. C. Freiling, and T. Dimitriou, “Cooperative intrusion detection in wireless sensor networks,” in *Proc. EWSN ’09*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 263–278.

[12] H. Lee, A. Cerpa, and P. Levis, “Improving wireless simulation through noise modeling,” in *Proc. IPSN ’07*. New York, NY, USA: ACM, 2007, pp. 21–30.

[13] T. Instruments, “2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver,” March 2007, http://focus.ti.com/lit/ds/symlink/cc2420.pdf (Retrieved: April 5, 2009).

[14] J. Polastre, J. Hill, and D. Culler, “Versatile low power media access for wireless sensor networks,” in *Proc. SenSys ’04*, 2004, pp. 95–107.

[15] A. Liu and P. Ning, “Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks,” in *Proc. IEEE IPSN ’08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 245–256.

[16] D. Sy and L. Bao, “Captra: coordinated packet traceback,” in *Proc. IPSN ’06*, 0-0 2006, pp. 152–159.

[17] S. Gupta, R. Zheng, and A. Cheng, “Andes: an anomaly detection system for wireless sensor networks,” in *Proc. IEEE MASS ’07*, Oct. 2007, pp. 1–9.

[18] L. Yu and J. Li, “Spymon: Hidden network monitoring for security in wireless sensor networks,” in *Proc. IEEE MASS ’08*, 29 2008-Oct. 2 2008, pp. 328–333.

[19] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens, “Odsbr: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks,” *ACM Trans. Inf. Syst. Secur.*, vol. 10, no. 4, pp. 1–35, 2008.