An On-Demand Secure Routing Protocol Resilient to Byzantine Failures

Baruch Awerbuch, David Holmer, Cristina Nita-Rotaru and Herbert Rubens
Department of Computer Science
Johns Hopkins University
3400 North Charles St.
Baltimore, MD 21218 USA
{baruch, dholmer, crisn, herb}@cs.jhu.edu

ABSTRACT

An ad hoc wireless network is an autonomous self-organizing system of mobile nodes connected by wireless links where nodes not in direct range can communicate via intermediate nodes. A common technique used in routing protocols for ad hoc wireless networks is to establish the routing paths on-demand, as opposed to continually maintaining a complete routing table. A significant concern in routing is the ability to function in the presence of byzantine failures which include nodes that drop, modify, or mis-route packets in an attempt to disrupt the routing service.

We propose an on-demand routing protocol for ad hoc wireless networks that provides resilience to byzantine failures caused by individual or colluding nodes. Our adaptive probing technique detects a malicious link after $\log n$ faults have occurred, where n is the length of the path. These links are then avoided by multiplicatively increasing their weights and by using an on-demand route discovery protocol that finds a least weight path to the destination.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection; C.2.1 [**Network Architecture and Design**]: Wireless communication; C.2.2 [**Network Protocols**]: Routing protocols

General Terms

Algorithms, Design, Reliability, Security, Theory

Keywords

ad hoc wireless networks, on-demand routing, security, byzantine failures

WiSe'02, September 28, 2002, Atlanta, Georgia, USA.

1. INTRODUCTION

Ad hoc wireless networks are self-organizing multi-hop wireless networks where all the hosts (nodes) take part in the process of forwarding packets. Ad hoc networks can easily be deployed since they do not require any fixed infrastructure, such as base stations or routers. Therefore, they are highly applicable to emergency deployments, natural disasters, military battle fields, and search and rescue missions.

A key component of ad hoc wireless networks is an efficient routing protocol, since all of the nodes in the network act as routers. Some of the challenges faced in ad hoc wireless networks include high mobility and constrained power resources. Consequently, ad hoc wireless routing protocols must converge quickly and use battery power efficiently. Traditional proactive routing protocols (link-state [1] and distance vectors [1]), which use periodic updates or beacons which trigger event based updates, are less suitable for ad hoc wireless networks because they constantly consume power throughout the network, regardless of the presence of network activity, and are not designed to track topology changes occurring at a high rate.

On-demand routing protocols [2, 3] are more appropriate for wireless environments because they initiate a route discovery process only when data packets need to be routed. Discovered routes are then cached until they go unused for a period of time, or break because the network topology changes.

Many of the security threats to ad hoc wireless routing protocols are similar to those of wired networks. For example, a malicious node may advertise false routing information, try to redirect routes, or perform a denial of service attack by engaging a node in resource consuming activities such as routing packets in a loop. Furthermore, due to their cooperative nature and the broadcast medium, ad hoc wireless networks are more vulnerable to attacks in practice [4].

Although one might assume that once authenticated, a node should be trusted, there are many scenarios where this is not appropriate. For example, when ad hoc networks are used in a public Internet access system (airports or conferences), users are authenticated by the Internet service provider, but this authentication does not imply trust between the individual users of the service. Also, mobile devices are easier to compromise because of reduced physical security, so complete trust should not be assumed.

Our contribution. We focus on providing routing survivability under an adversarial model where any intermediate node or group of nodes can perform byzantine attacks such as creating routing loops, misrouting packets on non-optimal paths, or selectively dropping packets (*black hole*). Only the source and destination nodes are assumed to be trusted. We propose an on-demand routing protocol for wireless ad hoc networks that operates under this strong adversarial model.

It is provably impossible under certain circumstances, for example when a majority of the nodes are malicious, to attribute a byzantine fault occurring along a path to a specific node, even using expensive and complex byzantine agreement. Our protocol circumvents this obstacle by avoiding the assignment of "guilt" to individual nodes. Instead it reduces the possible fault location to two adjacent nodes along a path, and attributes the fault to the link between them. As long as a fault-free path exists between two nodes, they can communicate reliably even if an overwhelming majority of the network acts in a byzantine manner.

Our protocol consists of the following phases:

- *Route discovery with fault avoidance.* Using flooding and a faulty link weight list, this phase finds a least weight path from the source to the destination.

- *Byzantine fault detection.* This phase discovers faulty links on the path from the source to the destination. Our adaptive probing technique identifies a faulty link after $\log n$ faults have occurred, where n is the length of the path.

- *Link weight management.* This phase maintains a weight list of links discovered by the fault detection algorithm. A multiplicative increase scheme is used to penalize links which are then rehabilitated over time. This list is used by the route discovery phase to avoid faulty paths.

The rest of the paper is organized as follows. Section 2 summarizes related work. We further define the problem we are addressing and the model we consider in Section 3. We then present our protocol in Section 4 and provide an analysis in Section 5. We conclude and suggest directions for future work in Section 6.

2. RELATED WORK

Secure routing protocols for ad hoc wireless networks is a fairly new topic. Although routing in ad hoc wireless networks has unique aspects, many of the security problems faced in ad hoc routing protocols are similar to those faced by wired networks. In this section, we review the work done in securing routing protocols for both ad hoc wireless and wired networks.

One of the problems addressed by researchers is providing an effective public key infrastructure in an ad hoc wireless environment which by nature is decentralized. Examples of these works are as follows. Hubaux et al.[5] proposed a completely decentralized public-key distribution system similar to PGP [6]. Zhou and Haas [7] explored threshold cryptography methods in a wireless environment. Brown et al.[8] showed how PGP, enhanced by employing elliptic curve cryptography, is a viable option for wireless constrained devices.

A more general trust model where levels of security are defined for paths carrying specific classes of traffic is suggested in [9]. The paper discusses very briefly some of the cryptographic techniques that can be used to secure on-demand routing protocols: shared key encryption associated with a security level and digital signatures for data source authentication.

As mentioned in [10], source authentication is more of a concern in routing than confidentiality. Papadimitratos and Haas showed in [11] how impersonation and replay attacks can be prevented for on-demand routing by disabling route caching and providing end-to-end authentication using an HMAC [12] primitive which relies on the existence of security associations between sources and destinations. Dahill et al.[16] focus on providing hop-by-hop authentication for the route discovery stage of two well-known on-demand protocols: AODV [2] and DSR [3], relying on digital signatures. Other significant works include SEAD [13] and Ariadne [4] that provide efficient secure solutions for the DSDV [14] and DSR [3] routing protocols, respectively. While SEAD uses one-way hash chains to provide authentication, Ariadne uses a variant of the Tesla [15] source authentication technique to achieve similar security goals.

Marti et al.[18] address a problem similar to the one we consider, survivability of the routing service when nodes selectively drop packets. They take advantage of the wireless cards promiscuous mode and have trusted nodes monitoring their neighbors. Links with an unreliable history are avoided in order to achieve robustness. Although the idea of using the promiscuous mode is interesting, this solution does not work well in multi-rate wireless networks because nodes might not hear their neighbors forwarding communication due to different modulations. In addition, this method is not robust against collaborating adversaries.

Also, relevant work has been done in the wired network community. Many researchers focused on securing classes of routing protocols such as link-state [10, 19, 20, 21] and distance-vector [22]. Others addressed in detail the security issues of well-known protocols such as OSPF [23] and BGP [24]. The problem of source authentication for routing protocols was explored using digital signatures [23] or symmetric cryptography based methods: hash chains [10], chains of one-time signatures [20] or HMAC [21]. Intrusion detection is another topic that researchers focused on, for generic link-state [25, 26] or OSPF [27].

Perlman [28] designed the Network-layer Protocol with Byzantine Robustness (NPBR) which addresses denial of service at the expense of flooding and digital signatures. The problem of byzantine nodes that simply drop packets (*black holes*) in wired networks is explored in [29, 30]. The approach in [29] is to use a number of trusted nodes to probe their neighbors, assuming a limited model and without discussing how probing packets are disguised from the adversary. A different technique, flow conservation, is used in [30]. Based on the observation that for a correct node the number of bytes entering a node should be equal to the number of bytes exiting the node (within a threshold), the authors suggest a scheme where nodes monitor the flow in the network. This is done by requiring each node to have a copy of the routing table of their neighbors and reporting the incoming and outgoing data. Although interesting, the scheme does not work when two or more adversarial nodes collude.

3. PROBLEM DEFINITION AND MODEL

In this section we discuss the network and security assumptions we make in this paper and present a more precise description of the problem we are addressing.

3.1 Network Model

This work relies on a few specific network assumptions. Our protocol requires bi-directional communication on all links in the network. This is also a requirement of most wireless MAC protocols, including 802.11 [31] and MACAW [32]. We focused on providing a secure routing protocol, which addresses threats to the ISO/OSI network layer. We do not specifically address attacks against lower layers. For example, the physical layer can be disrupted by jamming, and MAC protocols such as 802.11 can be disrupted by attacks using the special RTS/CTS packets. Though MAC protocols can detect packet corruption, we do not consider this a substitute for cryptographic integrity checks [33].

3.2 Security Model and Considered Attacks

In this work we consider only the source and the destination to be trusted. Nodes that can not be authenticated do not participate in the protocol, and are not trusted. Any intermediate node on the path between the source and destination can be authenticated and can participate in the protocol, but may exhibit byzantine behavior. The goal of our protocol is to detect byzantine behavior and avoid it. We define *byzantine behavior* as any action by an authenticated node that results in disruption or degradation of the routing service. We assume that an intermediate node can exhibit such behavior either alone or in collusion with other nodes. More generally, we use the term *fault* to refer to any disruption that causes significant loss or delay in the network. A fault can be caused by byzantine behavior, external adversaries, lower layer influences, and certain types of normal network behavior such as bursting traffic.

An adversary or group of adversaries can intercept, modify, or fabricate packets, create routing loops, drop packets selectively (often referred to as a *black hole*), artificially delay packets, route packets along non-optimal paths, or make a path look either longer or shorter than it is. All the above attacks result in disruption or degradation of the routing service. In addition, they can induce excess resource consumption which is particularly problematic in wireless networks.

There are strong attacks that our protocol can not prevent. One of these strong attacks, referred to as a *wormhole* [4], is where two attackers establish a path and tunnel packets from one to another. For example, the attackers can tunnel route request packets that can arrive faster than the normal route request flood. This may result in non-optimal adversarial controlled routing paths. Our protocol addresses this attack by treating the wormhole as a single link which will be avoided if it exhibits byzantine behavior, but does not prevent the wormhole formation. Also, we do not address traditional denial of service attacks which are characterized by packet injection with the goal of resource consumption.

Whenever possible, our protocol uses efficient cryptographic primitives. This requires pairwise shared keys[1] which are established on-demand. The public-key infrastructure used for authentication can be either completely distributed (as described in [5]), or Certificate Authority (CA) based. In the latter case, a distributed cluster of peer CAs sharing a common certificate and revocation list can be deployed to improve the CA's availability.

[1]We discourage group shared keys since this is an invitation for impersonation in a cooperative environment.

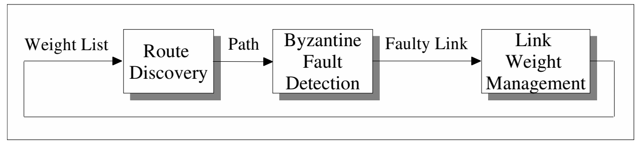

Figure 1: Secure Routing Protocol Phases

3.3 Problem Definition

The goal of this work is to provide a robust on-demand ad hoc routing service which is resilient to byzantine behavior and operates under the network and security models described in Sections 3.1 and 3.2. We attempt to bound the amount of damage an adversary or group of adversaries can cause to the network.

4. SECURE ROUTING PROTOCOL

Our protocol establishes a reliability metric based on past history and uses it to select the best path. The metric is represented by a list of link weights where high weights correspond to low reliability. Each node in the network maintains its own list, referred to as a *weight list*, and dynamically updates that list when it detects faults. Faulty links are identified using a secure adaptive probing technique that is embedded in the normal packet stream. These links are avoided using a secure route discovery protocol that incorporates the reliability metric.

More specifically, our routing protocol can be separated into three successive phases, each phase using as input the output from the previous (see Figure 1):

- *Route discovery with fault avoidance.* Using flooding, cryptographic primitives, and the source's weight list as input, this phase finds and outputs the full least weight path from the source to the destination.

- *Byzantine fault detection.* The goal of this phase is to discover faulty links on the path from the source to the destination. This phase takes as input the full path and outputs a faulty link. Our adaptive probing technique identifies a faulty link after $\log n$ faults occurred, where n is the length of the path. Cryptographic primitives and sequence numbers are used to protect the detection protocol from adversaries.

- *Link weight management.* This phase maintains a weight list of links discovered by the fault detection algorithm. A multiplicative increase scheme is used to penalize links which are then rehabilitated over time. The weight list is used by the route discovery phase to avoid faulty paths.

4.1 Route Discovery with Fault Avoidance

Our route discovery protocol floods both the route request and the response in order to ensure that if any fault free path exists in the network, a path can be established. However, there is no guarantee that the established path is free of

```
Procedure list:
CreateSignSend( item1, item2, ... ) - creates a message of the concatenated item list, signed by the current node, and broadcasts it
Broadcast( message ) - broadcasts a message
VerifySignature( node, signature ) - verifies the signature and exits the procedure if the signature is not valid
Find( list, item ) - returns an item in a list, or NULL if the item does not exist
InsertList( list, item ) - inserts an item in a list
UpdateList( list, item ) - replaces the item in a list
LinkWeight( weight_list, A, B ) - returns the listed weight of the link between A and B, or one if the link is not listed

Code executed at node source when a new route to node destination is needed:
CreateSignSend( REQUEST, destination, source, req_sequence, weight_list )

Code executed at node this_node when a request message req is received:
if( Find( requests_list, req) = NULL )
    VerifySignature( req.source, req.signature )
    if( this_node = req.destination )
        CreateSignSend( RESPONSE, req.destination, req.source, req.req _sequence, req.high_weights_list )
    else
        Broadcast( req )
    endif
    InsertList(requests_list, req)
endif

Code executed at node this_node when a response message res is received:
update = false
prev_node = res.destination
total_weight = 0
for( i = 0; i < res.no_hops; i++ )
    total_weight += LinkWeight( res.weight_list, prev_node, res.hops[i].node )
    prev_node = res.hops[i].node
endfor
res.total_weight = total_weight + LinkWeight( res.weight_list, prev_node, this_node )
prev_response = Find( responses_list, res )
if( prev_response ≠ NULL )
    if( res.total_weight ≥ prev_response.total_weight )
        update = true
    endif
else
    update = true
endif
if( update )
    VerifySignature( res.destination, res.signature )
    for( i = 0; i < res.no_hops; i++ )
        VerifySignature( res.hops[i].node, res.hops[i].signature )
    endfor
    if( this_node = source )
        UpdateList( path_list, res )
    else
        CreateSignSend( res, this_node )
        UpdateList( responses_list, res )
    endif
endif
```

Figure 2: Route Discovery Algorithm

adversarial nodes. The initial flood is required to guarantee that the route request reaches the destination. The response must also be flooded because if it was unicast, a single adversary could prevent the path from being established. If an adversary was able to prevent routes from being established, the fault detection algorithm would be unable to detect and avoid the faulty link since it requires a path as input in order to operate.

A digital signature is used to authenticate the source. This is required to prevent unauthorized nodes from initiating resource consuming route requests. An unauthorized route request would fail verification and be dropped by each of the requesting node's immediate neighbors, preventing the request from flooding through the network.

At the completion of the route discovery protocol, the source is provided with the complete path to the destination. Many on-demand routing protocols use route caching by intermediate nodes as an optimization; we do not consider it in this work because of the security implications. We intend to address route caching optimizations with strong security semantics in a future work.

Our route discovery protocol uses link weights to avoid faults. A weight list is provided by the link weight management phase (Section 4.3). The route discovery protocol chooses a route that is a minimum weight path between the source and the destination. This path is found during a flood by accumulating the cost hop by hop and forwarding the flood only if the new cost is less than the previously forwarded cost. The protocol uses digital signatures at each hop to prevent an adversary from specifying an arbitrary path. For example, it can stop an adversary from inventing a short path in an attempt to draw packets into a black hole. Since the cost associated with signing a message at each hop is very high, the weights are accumulated as part of the response flood instead of the request flood in order to minimize the cost of route requests to unreachable destinations.

If only the source verifies all of the weights and signatures, then the protocol becomes vulnerable to attacks on the response flood propagation. The adversaries could block correct information from reaching the source by propagating low cost fabricated responses. The source can ignore non-authentic responses, however, since intermediate nodes only re-send lower cost information, a valid response would never reach the source. Therefore, each intermediate node must verify the weights and the signatures carried by a response, in order to guarantee that a path will be established.

An adversary can still influence the path selection by creating what we refer to as *virtual links*. A virtual link is formed when adversaries form wormholes, as described in Section 3.2, or any other type of shortcuts in the network. A virtual link can be created by deleting one or more hops from the end of the route response. Our detection algorithm (Section 4.2) can identify and avoid virtual links if they exhibit byzantine behavior, but our route discovery algorithm does not prevent their formation. We present a detailed analysis of the effect of virtual links in Section 5.

As part of the route discovery protocol, each node maintains a list of recent requests and responses that it has already forwarded. The following five steps comprise the route discovery protocol (see also Figure 2):

I. Request Initiation. The source creates and signs a request that includes the destination, the source, a sequence number, and a weight list (see Line 1, Figure 2). The source then broadcasts this request to its neighbors. The source's signature allows the destination and intermediate nodes to authenticate the request and prevents an adversary from creating a false route request.

II. Request Propagation. The request propagates to the destination via flooding which is performed by the intermediate nodes as follows. When receiving a request, the node first checks its list of recently seen requests for a matching request (one with the exact same destination, source, and request identifiers). If there is no matching request in its list, and the source's signature is valid, it stores the request in its list and rebroadcasts the request (see Lines 2-10, Figure 2). If there is a matching request, the node does nothing.

III. Request Receipt / Response Initiation. Upon receiving a new request from a source for the first time, the destination verifies the authenticity of the request, creates and signs a response that contains the source, the destination, a response sequence number and the weight list from the request packet. The destination then broadcasts this response (see Lines 2-10, Figure 2).

IV. Response Propagation. When receiving a response, the node computes the total weight of the path by summing the weight of all the links on the specified path to this node (Lines 12-18, Figure 2). If the total weight is less than any previously forwarded matching response (same source, destination and response identifiers), the node verifies the signatures of the response header and every hop listed on the packet so far[2] (Lines 28-31, Figure 2). If the entire packet is verified, the node appends its identifier to the end of the packet, signs the appended packet, and broadcasts the modified response (Lines 35-36, Figure 2).

V. Response Receipt. When the source receives a response, it performs the same computation and verification as the intermediate nodes as described in the response propagation step. If the path in the response is better than the best path received so far, the source updates the route used to send packets to that specific destination (see Line 33, Figure 2).

4.2 Byzantine Fault Detection

Our detection algorithm is based on using acknowledgments (*acks*) of the data packets. If a valid ack is not received within a timeout, it is assumed that the packet has been lost. Note that this definition of loss includes both malicious and non-malicious causes. A loss can be caused by packet drop due to buffer overflow, packet corruption due to interference, a malicious attempt to modify the packet contents, or any other event that prevents either the packet or the ack from being received and verified within the timeout.

A network operating "normally" exhibits some amount of loss. We define a *threshold* that sets a bound on what is considered a tolerable loss rate. In a well behaved network the loss rate should stay below the threshold. We define a *fault* as a loss rate greater than or equal to the threshold.

[2] To maximize the performance of multiple verifications we use RSA keys with a low public exponent.

The value of the threshold also specifies the amount of loss that an adversary can create without being detected. Hence, the threshold should be chosen as low as possible, while still greater than the normal loss rate. The threshold value is determined by the source, and may be varied independently for each route to accommodate different situations, but this work uses a fixed threshold.

While this threshold scheme may seem overly "simple", we would like to emphasize that our protocol provides fault avoidance and never disconnects nodes from the network. Thus, the impact of false positives, due to normal events such as bursting traffic, is drastically reduced. This provides a much more flexible solution than one where nodes are declared faulty and excluded from the network. In addition, this avoidance property allows the threshold to be set very low, where it may be periodically triggered by false positives, without severely impacting network performance or affecting network connectivity.

A substantial advantage of our protocol is that it limits the overhead to a minimum under normal conditions. Only the destination is required to send an ack when no faults have occurred. If losses exceed the threshold, the protocol attempts to locate the faulty link. This is achieved by requiring a dynamic set of intermediate nodes, in addition to the destination node, to send acks to the source.

Normal topology changes occur frequently in ad hoc wireless networks. Although our detection protocol locates "faulty links" that are caused by these changes, an optimized mechanism for detecting them would decrease the overhead and detection time. Any of the mechanisms described in the route maintenance section of the DSR protocol [3], for instance MAC layer notification, can be used as an optimized topology change detector. When our protocol receives notification from such a detector, it reacts by creating a route error message that is propagated along the path back to the source. The node that generates this message, signs it, in order to provide integrity and authentication. Upon receipt of an authenticated route error message, the source passes the faulty link to the link weight management phase. Note that an intermediate node exhibiting byzantine behavior can always incriminate one of its links, so adding a mechanism that allows it to explicitly declare one of its links faulty, does not weaken the security model.

Fault Detection Overview. Our fault detection protocol requires the destination to return an ack to the source, for every successfully received data packet. The source keeps track of the number of recent losses (acks not received over a window of recent packets). If the number of recent losses violates the acceptable threshold, the protocol registers a fault between the source and the destination and starts a binary search on the path, in order to identify the faulty link. A simple example is illustrated in Figure 3.

The source controls the search by specifying a list of intermediate nodes on data packets. Each node in the list, in addition to the destination, must send an ack for the packet. We refer to the set of nodes required to send acks as probed nodes, or for short *probes*. Since the list of probed nodes is specified on legitimate traffic, an adversary is unable to drop traffic without also dropping the list of probed nodes and eventually being detected.

The list of probes defines a set of non-overlapping intervals that cover the whole path, where each interval covers the

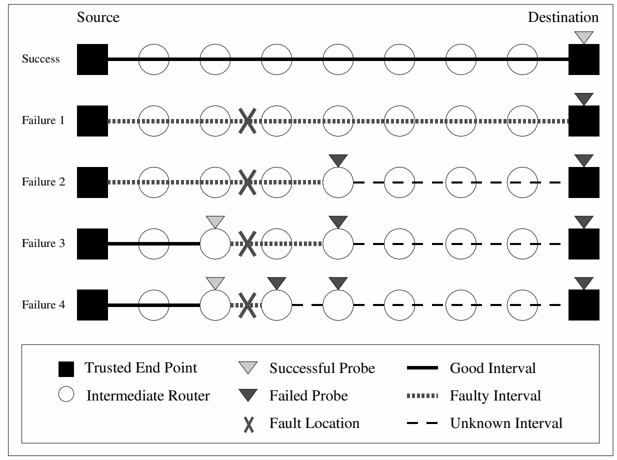

Figure 3: Byzantine Fault Detection

sub-path between the two consecutive probes that form its endpoints. When a fault is detected on an interval, the interval is divided in two by inserting a new probe. This new probe is added to the list of probes appended to future packets. The process of sub-division continues until a fault is detected on an interval that corresponds to a single link. In this case, the link is identified as being faulty and is passed as input to the link weight management phase (see Figure 1). The path sub-division process is a binary search that proceeds one step for each fault detected. This results in the identification of a faulty link after $\log n$ faults are detected, where n is the length of the path.

We use shared keys between the source and each probed node as a basis for our cryptographic primitives in order to avoid the prohibitively high cost of using public key cryptography on an per packet basis. These pairwise shared keys can be established on-demand via a key exchange protocol such as Diffie-Hellman [34], authenticated using digital signatures. The on-demand key exchange must be fully integrated into the fault detection protocol in order to maintain the security semantics. The integrated key exchange operates similarly to the probe and ack specification discussed below (see also Figure 4), but it is not described in detail in this work.

Probe Specification. The mechanism for specifying the probe list on a packet is essential for the correct operation of the detection protocol. The probes are specified in the list in the same order as they appear on the path. The list is "onion" encrypted [17]. Each probe is specified by the identifier of the node, an HMAC of the packet (not including the list), and the encrypted remaining list (see Lines 3-6, Figure 4). Both the HMAC and the encrypted remaining list are computed with the shared key between the source and that node. An HMAC [12] using a one-way hash function such as SHA1 [35] and a standard block cipher encryption algorithm such as AES [36] can be used.

A node can detect if it is required to send acks by checking the identifier at the beginning of the list (see Lines 8-12, Figure 4). If it matches, then it verifies the HMAC of the packet and replaces the list on the packet with the decrypted version of the remaining list. This mechanism forces the

Procedure list:
Cat(a, b, ...) - returns the concatenation of a, b, etc.
Hmac(data, key) - compute and return the hmac of data using key
Encrypt/Decrypt(data, key) - encrypt/decrypt data with key and return result
Report_Loss_and_Return(node) - reports that a loss was detected on the interval before *node* and exit the procedure

Code executed at source **when sending a packet with the contents** data **to** destination **:**

```
body = Cat( destination.id, source.id, destination.counter++, Encrypt ( data, destination.key ) )
tail = Hmac( body, destination.key )
for( i = probe_list.length - 1, i ≥ 0, i−− )
    tail = Encrypt( tail, probe_list[i].key )
    tail = Cat( probe_list[i].id, Hmac( body, probe_list[i].key ), tail )
endfor
Send( Cat( body, tail ) )
```

Code executed at this_node **when receiving a packet with the contents** source, destination, enc_data, id, hmac, enc_remainder **:**

```
if( id = this_node and hmac = Hmac( enc_data, source.key )
    Send( Cat( source, destination, enc_data, Decrypt( enc_remainder, key ) )
    waiting_for_ack = true
    Schedule_ack_timer()
endif
```

Code executed at destination **when receiving a packet with the contents** source, destination, counter, enc_data, hmac:

```
if( counter > prev_counter and hmac = Hmac( Cat( source, destination, counter, enc_data ) ) )
    Deliver( Decrypt( enc_data, source.key ) )
    Send( Cat( source.id, destination.id, counter, Hmac( Cat( destination.id, counter ), source.key ) ) )
endif
```

Code executed at probed_node **when receiving an ack with the contents** source, ack_node, counter, enc_remainder:

```
if( waiting_for_ack )
    encrypted_ack = Encrypt( Cat( ack_node, counter, enc_remainder ), source.key )
    Send( Cat( source.id, probed_node.id, counter, enc_ack, Hmac( Cat( probed_node.id, counter, enc_ack ), source.key ) ) )
    waiting_for_ack = false
    Unschedule_ack_timer()
endif
```

Code executed at this_node **when ack timer expires:**

```
waiting_for_ack = false
Send( Cat( source, this_node.id, counter, Hmac( Cat( this_node.id, counter ), source.key ) ) )
```

Code executed at source **when ack timer expires:**

```
waiting_for_ack = false
Report_Loss_and_Return( probe_list[0] )
```

Code executed at source **when receiving an ack with the contents** source, ack_node, counter, enc_remainder, hmac:

```
if( wait_for_ack and ack_node = probe_list[0].id and hmac = Hmac( Cat( ack_node, counter, enc_remainder ), ack_node.key ) )
    waiting_for_ack = false
    Unschedule_ack_timer()
    for( i = 1, i < probe_list.length, i++ )
        if( enc_remainder = NULL )
            Report_Loss_and_Return( probe_list[i] )
        endif
        ack_node, counter, enc_remainder, hmac = Decrypt( enc_remainder, probe_list[i-1].key )
        if( ack_node ≠ probe_list[i].id or hmac ≠ Hmac( Cat( ack_node, counter, enc_remainder ), ack_node.key ) )
            Report_Loss_and_Return( probe_list[i] )
        endif
    endfor
    if( enc_remainder = NULL )
        Report_Loss_and_Return( destination )
    endif
    ack_node, counter, hmac = Decrypt( enc_remainder, probe_list[i-1].key )
    if( ack_node ≠ destination or hmac ≠ Hmac( Cat( ack_node, counter ), destination.key ) )
        Report_Loss_and_Return( destination )
        return
    endif
    Success()
endif
```

Figure 4: Probe and Acknowledgement Specification

packet to traverse the probes in order, which verifies the route taken. Additionally, it verifies the contents of the packet at every probe point. The onion encryption prevents the adversary from incriminating other links by removing specific nodes from the probe list. Note that the adversary is able to remove the entire probe list, but this will incriminate one of its own links.

Acknowledgment Specification. If the adversary can drop individual acks, it can incriminate any arbitrary link along the path. In order to prevent this, each probe does not send its ack immediately, but waits for the ack from the next probe and combines them into one ack. Each ack consists of the identifier of the probe, the identifier of the data packet that is being acknowledged, the ack received from the next probe encrypted with the key shared by this probe and the source, and an HMAC of the new combined ack (see Lines 15 and 18-19, Figure 4).

If no ack is received within a timeout, the probe gives up waiting, and creates and sends its ack (see Line 24, Figure 4). The timeouts are set up in such a way that if there is a failure, all the acks before the failure point can be combined without other timeouts occurring. This is accomplished by setting the timeout for each probe to be the upper bound of the round-trip from it to the destination.

Upon receipt of an ack, the source checks the acks from each probe by successively verifying the HMACs and decrypting the next ack (see Lines 27-54, Figure 4). The source either verifies all the acks up through the destination, or discovers a loss on the interval following the last ack.

Interval and Probe Management. Let τ be the acceptable threshold loss rate. By using the above probe and acknowledgment specifications, it is simple to attribute losses to individual intervals. A loss is attributed to an interval between two probes when the source successfully received and verified an ack from the closer probe, but does not from the further probe. When the loss rate on an interval exceeds τ, the interval is divided in two.

Maintaining probes adds overhead to our protocol, so it is desirable to retire probes when they are no longer needed. The mechanism for deciding when to retire probes is based on the loss rate τ and the number of lost packets. The goal is to amortize the cost of the lost packets over enough good packets, so that the aggregate loss rate is bounded to τ.

Each interval has an associated counter C that specifies its lifetime. Initially, there is one interval with a counter of zero (there are initially no losses between the source and destination). When a fault is detected on an interval with a counter C, a new probe is inserted which *divides* the interval. Each of the two new intervals have their counters initialized to $\mu/\tau + C$, where μ is the number of losses that caused the fault. The counters are decremented for every ack that is successfully received, until they reach zero. When the counters of both intervals on either side of a probe reach zero, the probe is retired *joining* the two intervals.

In the worst case scenario, a dynamic adversary can cause enough loss to trigger a fault, then switch to causing loss just under τ in order to wait out the additional probe, and then repeat when the probe is removed. This results in a loss rate bounded to 2τ. If the adversary attempts to create a higher loss rate, the algorithm will be able to identify the faulty link.

4.3 Link Weight Management

An important aspect of our protocol is its ability to avoid faulty links in the process of route discovery by the use of link weights. The decision to identify a link as faulty is made by the detection phase of the protocol. The management scheme maintains the weight list using the history of faults that have been detected. When a link is identified as faulty, we use a multiplicative increase scheme to double its weight.

The technique we use for reseting a link weight is similar to the one we use for retiring probes (see Section 4.2). The weight of a link can be reset to half of the previous value after the counter associated with that link returns to zero. If μ is the number of packets dropped while identifying a faulty link, then the link's counter is increased by μ/τ where τ is the threshold loss rate. Each non-zero counter is reduced by $1/m$ for every successfully delivered packet, where m is the number of links with non-zero counters. This bounds the aggregate loss rate to 2τ in the worst case.

5. ANALYSIS

Our protocol ensures that, even in a highly adversarial controlled network, as long as there is one fault-free path, it will be discovered after a bounded number of faults have occurred. As defined in Section 4.2, a fault means a violation of the threshold loss rate. We consider a network of n nodes of which k exhibit adversarial behavior. The adversaries cooperate and create the maximum number of virtual links possible in order to slow the convergence of our algorithm.

We provide an analysis of the upper bound for the total number of packets lost while finding the fault free path. This bound is defined by the number of losses that result in an increase of the costs of all adversarial controlled paths above the cost of the fault free path.

Let q^- and q^+ be the total number of lost packets and successfully transmitted packets, respectively. Ideally, $q^- - \rho \cdot q^+ \leq 0$, where ρ is the transmission success rate, slightly higher than the original threshold. This means the number of lost packets is a ρ-fraction of the number of transmitted packets. While this is not quite true, it is true "up to an additive constant", i.e. ignoring a bounded number ϕ of packets lost. Specifically, we prove that there exists an upper bound ϕ for the previous expression. We show that:

$$q^- - \rho \cdot q^+ \leq \phi \tag{1}$$

Assume that there are k adversarial nodes, $k < n$. We denote by $\tilde{E}$ the set of "virtual links" controlled by adversarial nodes. The maximum size of $\tilde{E}$ is kn.

Consider a faulty link e, convicted j_e times and rehabilitated a_e times. Then, its weight, w_e, is at most n, $w_e = n$ means that the whole path is adversarial. By the algorithm, w_e is given by the formula:

$$w_e = 2^{j_e - a_e} \tag{2}$$

The number of convictions is at least $\frac{q^-}{\mu}$, so

$$\frac{q^-}{\mu} - \sum_{e \in \tilde{E}} j_e < 0. \tag{3}$$

Also, the number of rehabilitations is at most $\frac{q^+}{\mu/\rho}$, so

$$\sum_{e \in \tilde{E}} a_e - \frac{q^+}{\mu/\rho} < 0 \quad (4)$$

where μ is the number of lost packets that exposes a link. Thus

$$\frac{q^-}{\mu} - \frac{q^+}{\mu/\rho} \leq \sum_{e \in \tilde{E}} (j_e - a_e) \quad (5)$$

From Eq. 2 we have $j_e - a_e = \log w_e$. Therefore:

$$\sum_{e \in \tilde{E}} (j_e - a_e) = \sum_{e \in \tilde{E}} \log w_e \quad (6)$$

By combining Eq. 5 and 6, we obtain

$$q^- - \rho \cdot q^+ \leq \mu \sum_{e \in \tilde{E}} \log w_e \leq \mu \cdot kn \cdot \log n \quad (7)$$

and since $\mu = b \log n$, where b is the number of lost packets per window, Eq. 7 becomes

$$q^- - \rho \cdot q^+ \leq b \cdot kn \cdot \log^2 n \quad (8)$$

Therefore, the amount of disruption a dynamic adversary can cause to the network is bounded. Note that kn represents the number of links controlled by an adversary. If there is no adversarial node Eq. 8 becomes the ideal case where $q^- - \rho \cdot q^+ \leq 0$.

6. CONCLUSIONS AND FUTURE WORK

We presented a secure on-demand routing protocol resilient to byzantine failures. Our scheme detects malicious links after $\log n$ faults occurred, where n is the length of the routing path. These links are then avoided by the route discovery protocol. Our protocol bounds logarithmically the total amount of damage that can be caused by an attacker or group of attackers.

An important aspect of our protocol is the algorithm used to detect that a fault has occurred. However, it is difficult to design such a scheme that is resistant to a large number of adversaries. The method suggested in this paper uses a fixed threshold scheme. We intend to explore other methods, such as adaptive threshold or probabilistic schemes which may provide superior performance and flexibility.

In order to further enhance performance, we would like to investigate ways of taking advantage of route caching without breaching our security guarantees.

We also plan to evaluate the overhead of our protocol with respect to existing protocols, in normal, non-faulty conditions as well as in adversarial environments. Finally, we are interested in investigating means of protecting routing against traditional denial of service attacks.

Acknowledgments

We are grateful to Giuseppe Ateniese, Avi Rubin, Gene Tsudik and Moti Yung for their comments. We would like to thank Jonathan Stanton and Ciprian Tutu for helpful feedback and discussions. We also thank the anonymous referees for their comments.

We would like to thank the Johns Hopkins University Information Security Institute for providing the funding that made this research possible.

7. REFERENCES

[1] J. Kurose and K. Ross, *Computer Networking, a top down approach featuring the Internet.* Addison-Wesley Longman, 2000.

[2] C. E. Perkins and E. M. Royer, *Ad hoc Networking*, ch. Ad hoc On-Demand Distance Vector Routing. Addison-Wesley, 2000.

[3] D. B. Johnson, D. A. Maltz, and J. Broch, *DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. in Ad Hoc Networking*, ch. 5, pp. 139–172. Addison-Wesley, 2001.

[4] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," in *The 8th ACM International Conference on Mobile Computing and Networking*, September 2002. To appear.

[5] J.-P. Hubaux, L. Buttyan, and S. Capkun, "The quest for security in mobile ad hoc networks," in *The 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing*, October 2001.

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

[7] L. Zhou and Z. Haas, "Securing ad hoc networks," *IEEE Network Magazine*, vol. 13, November/December 1999.

[8] M. Brown, D. Cheung, D. Hankerson, J. Hernandez, M. Kirkup, and A. Menezes., "PGP in constrained wireless devices," in *The 9th USENIX Security Symposium*, USENIX, August 2000.

[9] S. Yi, P. Naldurg, and R. Kravets, "Security-aware ad hoc routing for wireless networks," in *The 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing*, October 2001.

[10] R. Hauser, T. Przygienda, , and G. Tsudik, "Reducing the cost of security in link-state routing," in *Symposium of Network and Distributed Systems Security*, 1997.

[11] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," in *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, pp. 27–31, January 2002.

[12] *The Keyed-Hash Message Authentication Code (HMAC).* No. FIPS 198, National Institute for Standards and Technology (NIST), 2002. http://csrc.nist.gov/publications/fips/index.html.

[13] Y.-C. Hu, D. B. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," in *The 4th IEEE Workshop on Mobile Computing Systems and Applications*, IEEE, June 2002.

[14] 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*, 1994.

[15] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Efficient and secure source authentication for multicast," in *Network and Distributed System Security Symposium*, February 2001.

[16] B. Dahill, B. Levine, C. Shields, and E. Royer, "A secure routing protocol for ad hoc networks," Tech. Rep. 01-37, Department of Computer Science,

University of Massachusetts, August 2001.

[17] P. F. Syverson, D. M. Goldschlag, and M. G. Reed, "Anonymous connections and onion routing," in *IEEE Symposium on Security and Privacy*, 1997.

[18] S. Marti, T. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *The 6th ACM International Conference on Mobile Computing and Networking*, August 2000.

[19] S. Cheung, "An efficient message authentication scheme for link state routing," in *The 13th Annual Computer Security Applications Conference*, pp. 90–98, December 1997.

[20] K. Zhang, "Efficient protocols for signing routing messages," in *Symposium on Networks and Distributed Systems Security*, 1998.

[21] M. T. Goodrich, "Efficient and secure network routing algorithms." Provisional patent filing., January 2001.

[22] B. R. Smith, S. Murthy, and J. Garcia-Luna-Aceves, "Securing distance-vector routing protocols," in *Symposium on Networks and Distributed Systems Security*, 1997.

[23] S. L. Murphy and M. R. Badger, "Digital signature protection of the OSPF routing protocol," in *Symposium on Networks and Distributed Systems Security*, 1996.

[24] B. Smith and J. Garcia-Luna-Aceves, "Efficient security mechanisms for the border gateway routing protocol," *Computer Communications (Elsevier)*, vol. 21, no. 3, pp. 203–210, 1998.

[25] S. F. Wu, F. yi Wang, B. M. Vetter, W. R. Cleaveland, Y. F. Jou, F. Gong, and C. Sargor, "Intrusion detection for link-state routing protocols," in *IEEE Symposium on Security and Privacy*, 1997.

[26] 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 *IEEE Symposium on Security and Privacy (5 Minutes)*, May 1997.

[27] S. Wu, H. Chang, D. Qu, F. W. F. Jou, F. Gong, C. Sargor, and R. Cleaveland, "JiNao: Design and implementation of a scalable intrusion detection system for the OSPF routing protocol," *Journal of Computer Networks and ISDN Systems*, 1999.

[28] R. Perlman, *Network Layer Protocols with Byzantine Robustness*. PhD thesis, MIT LCS TR-429, October 1988.

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

[30] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson, "Detecting disruptive routers: A distributed network monitoring approach," in *IEEE Symposium on Security and Privacy*, 1998.

[31] *ANSI/IEEE Std 802.11, 1999 Edition*. 1999. http://standards.ieee.org/catalog/olis/lanman.html.

[32] V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang, "MACAW: A media access protocol for wireless LAN's," in *SIGCOMM*, pp. 212–225, 1994.

[33] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," in *ACM SIGCOM*, August/September 2000.

[34] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644–654, November 1976.

[35] *Secure Hash Standard (SHA1)*. No. FIPS 180-1, National Institute for Standards and Technology (NIST), 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[36] *Advanced Encryption Standard (AES)*. No. FIPS 197, National Institute for Standards and Technology (NIST), 2001. http://csrc.nist.gov/encryption/aes/.