Optimistic Tracing in MANET *

Mike Burmester
Florida State University
Dept. of Computer Science
Tallahassee, FL 32306
burmester@cs.fsu.edu

Tri van Le
Florida State University
Dept. of Computer Science
Tallahassee, FL 32306
levan@cs.fsu.edu

March 28, 2006

Abstract: Mobile ad hoc networks are collections of wireless mobile nodes with links that are made or broken in an arbitrary way. They have constrained resources, restricted broadcast range and no fixed infrastructure. For these networks communication is achieved via routes whose nodes relay packets. Several routing algorithms have been proposed in the literature. These focus mainly on efficiency with security relegated to weak adversary models, particularly with regard to fault tolerance in malicious settings. Our goal in this paper is to develop a formal security framework for analyzing fault-tolerant routing in malicious environments. Our simulation framework allows for strong adversary models and preserves fault-tolerance in the universal composability paradigm. We present optimistic tracing protocols and prove their security in this model and show how they can be composed with existing, well-established routing algorithms. To the best of our knowledge, these are the only protocols that guarantee message delivery in a strong adversary model.

Keywords: MANET, Optimistic, Tracing, Secure, Routing.

1 Introduction

Finding and maintaining communication routes in mobile ad hoc networks is a major challenge, especially with respect to fault tolerance and security. To date, most of the research has focused on performance and services (see *e.g.*, [3, 16, 21, 22]) with security being given a lower priority, and in many cases, regarded as an add-on afterthought technology rather than a design feature (*e.g.*, [1, 20]). Although such an approach may be appropriate for networks with predictable faults, it is not suitable for networks with unpredictable, malicious faults. In particular one cannot trace malicious behavior by exploiting only stochastic network aspects, because malicious nodes may avoid detection by colluding and behaving normally whenever a fault detection mechanism is triggered. Of particular concern in military applications is the possibility that an established route is taken over by the adversary, and then used at a critical time. Another concern is that, besides packet dropping, malicious nodes may render a network useless by disseminating confusing information regarding the state of the system, *e.g.*, by blaming non-faulty nodes for failures and for dropping or corrupting packets. It is therefore important to trace malicious behavior and to prevent faulty nodes from taking part in future attacks.

In this paper we consider the problem of secure routing in mobile ad hoc networks when there are malicious faults. We overview current security threats and discuss countermeasures, focusing on routing issues. Our main contribution is to propose a formal security simulation framework for multi-party protocols, in the universal composability paradigm. We present two protocols that address malicious behavior and prove their security in our framework. Finally we show how these protocols can be combined to support the security of existing routing algorithms.

The paper organized as follows. In Section 2 we present our model that captures at an appropriate degree of abstraction the basic stochastic aspects of mobile ad hoc networks and give our definitions. In Section 3 we

*This material is based on work supported in part by the U.S. Army Research Laboratory and the U.S. Research Office under grant number DAAD19-02-1-0235 and in part by the National Science Foundation under grant number NSF-009316.

overview the security threats of routing algorithms. In Section 4 we present an algorithm that traces malicious faults and in Section 5 we present an adaptive multipath routing algorithm that tolerates malicious behavior.

2 Formal framework for security

Network: There are several ways to capture the unpredictable nature of a mobile ad hoc network. Whichever way is used, there are important mobility and medium aspects that must be reflected. Below we define a simple model for ad hoc network that will be assumed in our security framework.

Definition 1 A *mobile ad hoc network* is a stochastic process $\mathcal{G} = \mathcal{G}_1, \mathcal{G}_2, \ldots$, where $\mathcal{G}_t$ is a random graph with node set V, for which communication is: (i) *synchronous*, the time for a single transmission to be received is bounded by a constant; (ii) *promiscuous*, a packet transmitted by a node will be received by all its neighbors; (ii) *ordered*, packets transmitted by a node will be received at each of its neighbors in the same order that they were sent.

Our model allows the following to happen: (i) undeterministic arrival time, a packet can arrive at an arbitrarily future time within the synchronous bound; (ii) promiscuous constraint is only effective on mobile nodes, an adversary can listen to all packet transmissions can transmit packets to arbitrary nodes, regardless of the network structure; (iii) undeterministic arrival order among senders, packets transmitted by multiple nodes can arrive at any single receiver in an arbitrary order; (iv) underterministic arrival order among receivers, a packet transmitted by a node can arrive at its receivers in an arbitrary order.

Network links can be undirected (the neighbor relationship is symmetric) or directed (the neighbor relationship is asymmetric). We note that to the best of our knowledge, to date, all routing algorithms proposed in the literature support only bidirectional or undirected links.

The stochastic aspect of $\mathcal{G}$ is determined by the states of the nodes of $\mathcal{G}$ (including nodes under control of the adversary) and Nature. Nature's contribution comes from the environment and the fact that the communication is wireless. A wide variety of factors may affect the communication, ranging from weather to radio interference and physical obstacles.

Adversary structure: We allow the adversary to send packets to *arbitrary* nodes and to eavesdrop on *all* communication regardless of the network structure and can request arbitrary actions at nodes under his control. The adversary also has full information on the network connectivity. In these respects, our model is stronger than other models, in particular, the Byzantine model described below.

Definition 2 Let Γ be a family of subsets V' of the node set V. We call Γ an *Adversary Structure* [11]. The adversary $\mathcal{A} = \mathcal{A}_\Gamma$ selects a subset $V' \in \Gamma$ and can corrupt all of its nodes during the lifetime of the system.[1] $\mathcal{A}dv$ controls the nodes of V' and may use them to undermine the security of the network. We call these nodes *corrupted* or *faulty* and refer to $\mathcal{A}dv$ as a Γ-adversary. The adversary may be *passive* or *active*. A *passive* adversary (also called *honest-but-curious*) will only eavesdrop on the network communication. An *active* adversary may use the corrupted nodes to prevent the normal functioning of the network via *dropping*, *modifying*, and/or *fabricating* network messages. Nodes that are actively involved in such attacks and the corresponding faults are called *malicious* or *Byzantine*. Malicious nodes may use hidden (covert) channels or "wormholes" through which they can communicate or tunnel packets.

A particular case of the Adversary Structure model is the *Byzantine faults* model [24] for which $\Gamma = \{V' \subset V \mid |V'| \leq k\}$, for some threshold k. In this case we call $\mathcal{A} = \mathcal{A}_k$ a k-adversary.

In Figure 1: π is the protocol, $\mathcal{G}$ is the network graph, F_π the functionality that emulates π, $\mathcal{A}$ the adversary, $S_\mathcal{A}$ a simulator for the adversary.

Simulation Framework. Our security simulation framework follows cryptographic simulation paradigms for the security of network protocols [2]. Mobile nodes are probabilistic Turing machines with special tapes (transceiver) for sending and receiving packets.

[1] There are several generalizations of this model. One such generalization allows Γ to be dynamic: at regular intervals $\mathcal{A}dv$ can replace V' by $V'' \in \Gamma$, that is, release the nodes of $V' \backslash V''$ and replace them by the nodes of $V'' \backslash V'$. Another generalization involves hybrid faults: malicious faults and physical faults. We shall not consider these models here.

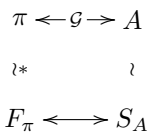

Figure 1: The UC security framework.

We allow the adversary is to schedule the delivery of network packets and interleave the delivery with its own packets in arbitrary way it wishes, subject only to the synchrony, promiscuity and orderly conditions on the part of the network packets. In other words, we assume that the whole network is run by the adversary.

For concurrent executions of a distributed algorithm π, we use a model with an infinite collection of oracles $\{\mathcal{O}_\pi^\sigma \mid \text{session } \sigma\}$ that simulate concurrent sessions of π. The adversary can interact concurrently with the oracles. In particular, in session σ, the adversary is able to interact with uncorrupted mobile nodes by querying oracle $\mathcal{O}_\pi^\sigma$. Each query to $\mathcal{O}_\pi^\sigma$ emulates one round of interaction in the session. A query to oracle $\mathcal{O}_\pi^\sigma$ is a list of packets to be written to the receive tape of each mobile node. $\mathcal{O}_\pi^\sigma$ first combines this input with the list of packets to be delivered—which was determined in the previous round—into a single list; it then delivers the packets (from the combined list) to the corresponding nodes and simulates the response of each node according to protocol π; finally, it forwards the nodes' responses to both (i) the oracle $\mathcal{N}_\mathcal{G}$, whose output determines the list of packets to be delivered to mobile nodes in the next round; and (ii) to the adversary as $\mathcal{O}_\pi^\sigma$'s output.

The view $view_\pi^\sigma$ of protocol π in session σ is the list of all packets sent and received by uncorrupted mobile nodes in session σ. This view describes the execution of the protocol π in session σ under the influence of the adversary. Let $success(view_\pi^\sigma) = 1$ if session σ is a successful execution of protocol π and $success(view_\pi^\sigma) = 0$ otherwise.

Definition 3 Let π be a distributed algorithm that runs in at most $t(\pi)$ rounds and $\mathcal{A}^{\mathcal{N}_\mathcal{G}, \mathcal{O}_\pi^\sigma}$ be an Γ-adversary with oracle accesses to $\mathcal{N}_\mathcal{G}$ and $\{\mathcal{O}_\pi^\sigma\}$. Let $\Sigma_\pi^\mathcal{A}$ be the set of sessions σ queried by adversary $\mathcal{A}^{\mathcal{N}_\mathcal{G}, \mathcal{O}_\pi^\sigma}$ at least $t(\pi)$ times during its execution. We define the advantage of $\mathcal{A}^{\mathcal{N}_\mathcal{G}, \mathcal{O}_\pi^\sigma}$ against π by:

$$adv_\pi^\mathcal{A} = \text{Prob}[\exists \sigma \in \Sigma_\pi^\mathcal{A} : success(view_\pi^\sigma) = 0].$$

The probabilities are taken over the adversary's and honest parties' coin tosses. The following definition captures the basic availability requirements of this approach for secure distributed applications.

Definition 4 We have:

- Γ-*availability* for π if $adv_\pi^\mathcal{A}$ is negligible for all Γ-adversaries $\mathcal{A}$.
- Γ-*tolerance* for π if $|adv_\pi^\mathcal{A} - adv_\pi^\emptyset|$ is negligible for all Γ-adversaries $\mathcal{A}$.

Here negligibility is defined as usual [9] and $\emptyset$ is the passive adversary that only eavesdrops.

The above definition covers availability and tolerance in a multi-party setting.

2.1 Security notations and mechanisms

For data integrity, Message Authentication Codes (MACs) may be used. For authenticity and integrity, digital signatures are used. For confidentiality (privacy) encryption mechanisms are used [24]. These are all keyed cryptosystems. There are two types of cryptosystems: *symmetric* and *public key*. Symmetric cryptosystems require one shared *secret* key. Public key cryptosystems require two keys, a *public key* and a *secret key*. In our algorithms we shall use the following notation:

- $[[data]]_{sd}$: $data$, and its keyed MAC with the shared key of s, d.
- $[data]_x$: $data$, and its digital signature with the signing key of x.
- $hash(data)$: the (cryptographic) hash of $data$ [24].

We assume in this paper that all MACs and digital signatures are unforgeable. In particular, that the network nodes and the adversary are polynomially bounded in the security parameter of the signatures. Consequently, the security is conditional, and the error probability must take into account the error probabilities of these cryptographic mechanisms.

We describe our optimistic tracing protocol using public key digital signature. However, symmetric key mechanisms such as hash chain and TESLA [25] can also be used interchangably for this purpose, which requires more involved key predistribution, nevertheless allows substantially faster signing and verification.

We shall assume that each network node is assigned a unique secret signing key and given a list of public keys that correspond to the assigned secret keys. This will allow nodes to link digitally signed messages to their owners and to authenticate nodes. It is important however to note that malicious nodes may choose to share their secret signing keys. This will make it possible for them to appear to be present in several virtual places of the network at the same time (this is the Sybil attack [5] which we shall discuss in Section 3.2). We therefore view malicious nodes as collections of virtual nodes, each one corresponding to a unique signing key.

The inability to bind entities (or messages) to a unique *physical node* is an inherent limit of Public Key Cryptography. It is not restricted to networks and applies to all protocols that rely on cryptographic primitives for authentication.

3 Security issues for routing algorithms

Depending on where most of the routing effort takes place, there are two types of routing: *network-centric* and *source-centric*. With network-centric routing (such as DSDV [21], WR [3] and AODV [22]) the routing effort is distributed within the network; with source-centric routing (such as DSR [16]) most of the routing effort is done by the source node.

Network-centric routing requires considerable cooperation between the nodes of the network in order to update and maintain a distributed database of routing information such as routes, cost, distance, reliability, time, etc. This type of routing is appropriate for networks whose node mobility is low and changes are less frequent. Its advantage is that the routing service is always available and communication can start almost immediately. ¿From a security point of view, network-centric routing requires substantial cooperation between network nodes and strong trust relationships. These algorithms are therefore more vulnerable to malicious faults. There is no way to prevent such faults, because the routing service is provided by remote nodes (that may be faulty).

With source-centric routing, the source s is responsible for discovering the topology of the network, for finding a route and for updating any changes, with less help from other nodes. When a node needs to send a packet, a route to the destination is constructed on-demand by the node and updated according to the changes in the network. Cooperation from other nodes is often limited to forwarding packets or collecting local information. Since there is almost no status information to maintain, this kind of routing is flexible and appropriate for networks that change frequently. Source-centric routing lessens the dependence on intermediate node cooperation, and thus is less vulnerable to malicious attacks. Furthermore, since the source and destination have control over the routes, they are also more flexible in dealing with DoS. For these reasons, when security issues are of concern, source-centric routing is preferable.

3.1 Denial of Service attacks and countermeasures

There are several ways in which a DoS can be triggered. For example, the adversary can cause a DoS by flooding the network with irrelevant packets (via faulty nodes). Another way to trigger a DoS is by flooding queries in dense networks. We also have DoS attacks on routes. If the adversary succeeds in taking control of a route, for example by having one or more nodes under his control selected by a route discovery algorithm, then the adversary will establish routes that may not exist or that may have loops, which could prevent routing updates from settling and route convergence. DoS is also triggered by packet dropping. For example, malicious nodes in a route discovery algorithm may drop packets to prevent the source getting path information. Packet dropping can also take place during communication. This problem is aggravated when malicious nodes collude.

Non-malicious DoS caused by flooding in dense networks is controlled by reducing the broadcast redundancy. *Gossip* protocols [10, 4] use this approach. Malicious DoS caused by flooding may be controlled by using Intrusion Detection mechanisms. One way to deal with malicious DoS attacks on routes is to use fault tracing algorithms. Awerbuch-Holmer-Nita Rotaru-Rubens [1] use an adaptive fault probing algorithm that is triggered when faults

occur at a rate higher than that of ordinary link failures (non-malicious). There are several problems with such an approach, due primarily to the fact that a malicious node need not exhibit faulty behavior when probed, but only during communication. Furthermore, malicious nodes may collude to prevent failure reports reaching the source and make bogus reports to confuse other nodes. In Section 4 we describe an algorithm that will trace malicious behavior when it occurs.

3.2 Man-in-the-Middle attacks and countermeasures

In a man-in-the-middle attack the adversary takes control of the communication channel between the source and destination by interposing between them. In their simplest form these attacks are *passive*, with the adversary relaying packets between two nodes x, y via nodes under his control. The relaying node(s) is (are) transparent to x and y, and x is fooled into believing that y is in range (a neighbor). In particular x, y will appear to be adjacent in any route containing them. The attacker will not be listed on the route, but the nodes x, y will be. Consequently, the route will appear to be shorter than it actually is, and may be selected in preference to other routes. In this way the adversary can take control of the route. Authentication mechanisms are of no help: the adversary simply relays the authenticators.

Active man-in-the-middle attacks in which the attacker is an "insider", that is a malicious node that is trusted, are the hardest to control. In such attacks, the attacker is properly authenticated and controls nodes on routes originating at the source. In a *wormhole attack* [14] the adversary succeeds in fooling a source node into believing that a route is short by tunnelling packets intended for the destination via nodes under her control. A *rushing attack* [14] is a wormhole attack in which the adversary succeeds in sending packets through the wormhole faster than normal network traffic. With such attacks it may not possible to distinguish non-faulty nodes from malicious nodes because the adversary may disguise the attack to mimic (stochastically) a failure caused by Nature. In a *Sybil attack* [5] a malicious node z presents multiple identities. In this way z succeeds in fooling the source into believing that there are many short routes to the destination. These routes "pass through" conspiring nodes z_i that may actually be far away (in broadcast hops), but are used as proxy nodes by the nearby node z. In this attack z knows the secret authentication keys of the conspiring nodes z_i and uses them to authenticate the z_i.

Man-in-the-middle attacks in ad hoc networks are hard to counter, if not impossible. There are two general approaches that can be used with such attacks: a *temporal* and a *locational* approach. The former exploits the time taken for each broadcast hop. In most cases this can be used to prevent the attacker from falsifying the length of routes. The latter uses the physical location of the nodes. Each node certifies its own position. In most cases this approach will trace nodes that claim false positions (by non-faulty neighbor nodes).

3.3 Security at the physical and data link layers

There are two types of faults that may occur in a routing algorithm: faults whose effect is stochastically indistinguishable from ordinary link failures caused by the mobility of the system, radio interference, power failure etc, and faults whose effect can be distinguished. Malicious faults tend to be of the second type, although the first type should not be excluded. For example, as observed earlier, the adversary may try to evade detection by causing faults that mimic the statistics of natural failures. Furthermore, malicious physical faults may affect the mobility of the system.

Faults that deviate from ordinary failures can be controlled by using redundancy. In particular, error detection, error correction and erasure mechanisms. These faults are best dealt with at the physical or data link layer of the protocol stack with Medium Access Control protocols. At these layers one can also deal with jamming attacks (using frequency-hopping spread spectrum techniques) and most isolated DoS attacks.

Faults of the second type, although by definition statistically detectable, can be quite hard to trace or locate. They include malicious faults. Malicious faults may occur when they are least expected, and may not be traceable with statistical failure analysis. The reason for this is that any analysis based on reported failures can be manipulated by the adversary. Faults of this type have to be addressed at the network layer. In this paper we are concerned with such faults.

3.4 Security issues of Ariadne, SEAD and SAODV

Several routing protocols in the literature address security issues (see *e.g.*, [20]). Here we discuss three of the more popular ones: Ariadne [12], SEAD [13] and Secure AODV [28].

Ariadne is a source-centric routing algorithm based on DSR that uses an authentication mechanism with a keyed hash chain called TESLA [25] for path integrity. The security of this algorithm is based on the assumption that all nodes on a route (insiders) will protect the integrity of path information. It therefore will not tolerate insider faults. In particular it does not tolerate DoS caused by packet dropping. SEAD is a source-centric variant of Ariadne. This algorithm also does not tolerate insider faults. Secure AODV (SAODV) is a network-centric routing algorithm that is based on the AODV algorithm [22]. It uses digital signatures and hash chains to protect the integrity of path information. As with the previous two algorithms it will not tolerate insider faults.

Rushing attacks on routing algorithms are the hardest to control. With these attacks two colluding nodes, one close to the source s the other close to the destination d, tunnel packets intended for d and sent by s via a wormhole, slightly faster than normal network traffic. The colluding nodes are authenticated and may insert conspiring nodes (using a Sybil attack) on the path to make its length appear "normal" and be selected in preference to other paths. Such attacks are not tolerated by Ariadne, SEAD and SAODV.

4 Tracing malicious faults

In this section we describe a routing algorithm that will trace malicious faults by identifying malicious behavior. Faulty nodes that are traced may have their keys invalidated by the non-faulty nodes, thus preventing future attacks.

Observe that failure rates based on reported failures of nodes to forward packets may be inaccurate. This is because faulty nodes may fail to report such events –even worse, fabricate events. Consequently tracing mechanisms that are triggered by failure rates exceeding a certain threshold may fail. Furthermore it is not possible in general to tell from a report by a node that claims that another node is faulty, which node is actually faulty: the reporting or the reported node. Two approaches can be used with malicious k-adversaries. In the first, malicious behavior is established when more than k (distinct) reports are available. In the second, each time a node is reported as malicious, both the reporting and the reported node are treated as malicious and eliminated. In this case the malicious nodes can cause up to k faults, but will then be eliminated together with up to k non-faulty nodes.

4.1 An optimistic algorithm that traces malicious faults

We describe an optimistic[2] algorithm that will trace malicious node behavior. For this algorithm there is no additional cost when there are no faults. When faults do occur, the cost to locate a fault is one tracing round and one digital signature. Compared to [1], our algorithm will locate faults when malicious nodes collude and it also uses less rounds. Each participating node only needs to know its neighbors on the path. In this algorithm faults that can be dealt with at the data link layer by error correction and re-sending packets are treated as non-malicious. The protocol is described in Figure 2. We use the following notation:

- $pkt_{sd} = [[s, d, sn, seq_s, data]]_{sd}$: an authenticated packet consisting of identifiers s, d, a session number sn for tracing algorithm (unique to each session), the sequence number seq_s for pkt_s, and $data$.
- $ack_{sd} = [[s, d, sn, seq_s]]_{sd}$: an authenticated acknowledgment.
- $prob_s = [s, d, sn, seq_s, hash(pkt_s)]_s$: a digitally signed probing request by s.
- $failreport_y = [s, d, y, succ(y), sn, seq_s]_y$: a digitally signed failure report by y.
- $timer_{xy}$: a bound on time taken for a round trip from x to y for pkt_s.
- $prec(x)$, $succ(x)$: the node that precedes, succeeds x on the path taken by pkt_s.

In the protocol, the source s sends a packet pkt_s to $succ(s)$ to be delivered to the destination d. If there are no faults then the packet reaches d that will send back to s an authenticated acknowledgment ack_d. If there is a fault and this is detected by an intermediate nodes y, then a $failreport_y$ will be sent to s. Otherwise the source s will send a $prob_s$ with details of ack_d requesting from intermediate nodes to check the validity of any received $failreport_y$ or ack_d. Thus, for an intermediate node x, either $succ(x)$ is faulty or x should have received from $succ(x)$,

1. after $x \xrightarrow{pkt_s} succ(x)$ and before $timer_{xd}$ timeouts: either an ack_d for which (s, d, sn, seq_s) have the correct values, or

[2]Optimistic algorithms have optimal performance when they are no faults.

2. after $x \xrightarrow{prob_s} succ(x)$ and before the reset $timer_{xd}$ timeouts: a valid $failreport_y$.

It follows that s will receive a valid $failreport_y$ and consequently a fault will be traced. Observe that in the protocol s, d check the validity of pkt_s and ack_d, and if there are no faults, the intermediate nodes check only for matching acknowledgments ack_d; if there are faults, intermediate nodes will also check the validity of $failreport_y$ and $prob_s$.

Source s**.** Set $seq_s = 0$. While a connection to d has not terminated do:

1. Set $timer_{sd}$ and send pkt_{sd} to $succ(s)$.
2. If a valid ack_{sd} for pkt_{sd} is received before timeout then set $seq_s = seq_s + 1$.
3. Else reset $timer_{sd}$ and send $prob_s$ to $succ(s)$.
   (a) If a valid $failreport_y$ for pkt_{sd} is received before timeout then y or $succ(y)$ is malicious;
   (b) Else $succ(s)$ is malicious.

Intermediate node x**.** When pkt'_{sd} is received:

1. Set $timer_{xd}$ and $timer_{xs}$, and send pkt'_{sd} to $succ(x)$.
2. If an ack'_{sd} matching pkt'_{sd} is received before $timer_{xd}$ timeouts then send ack'_{sd} to $prec(s)$.
   (a) If a valid $probe_s$ for pkt'_{sd} is received with $ack_{sd} \neq ack'_{sd}$ before $timer_{xs}$ timeouts, reset $timer_{xd}$ and send $probe_s$ to $succ(x)$.
      i. If a valid $failreport_y$ for pkt'_{sd} is received before $timer_{xd}$ timeouts:
      send $failreport_y$ to $prec(x)$;
      ii. Else construct and send $failreport_x$ to $prec(x)$.
3. Else if a valid $probe_s$ for pkt'_{sd} is received with $pkt_{sd} = pkt'_{sd}$ before $timer_{xs}$ timeouts, reset $timer_{xd}$ and send $probe_s$ to $succ(x)$.
   (a) If a valid $failreport_y$ for pkt'_{sd} is received before $timer_{xd}$ timeouts: send $failreport_y$ to $prec(x)$;
   (b) Else construct and send $failreport_x$ to $prec(x)$.

Destination d**.** When a valid pkt_s is received:

1. Construct and send ack_d to $prec(d)$.

Figure 2: An optimistic tracing algorithm.

Theorem 1 *For any Γ-adversary, the tracing algorithm in Figure 2 will either deliver pkt_s to the destination d or will trace at least one faulty node. In particular:*

1. *If all nodes adhere to the protocol then d will receive pkt_{sd} and the source s will receive ack_{sd} before its timeout.*
2. *If s received a valid ack_{sd} before its timeout then d will have received pkt_{sd}.*
3. *If s did not receive a valid ack_{sd} before its timeout then at least one faulty node is traced.*

Proof. (Sketch) We consider each part separately.

1. Clearly if all nodes adhere to the protocol then d will get pkt_{sd} and s will get ack_{sd}.

2. If s gets a valid ack_{sd}, then because we assume that digital signatures are unforgeable and because d will only sign a matching ack_{sd} if the received pkt_{sd} is valid, d must have received pkt_{sd}.

3. If s has not received a valid ack_{sd} before its timeout, it will send a probe $prob_s$ downstream requesting intermediate nodes to check the last transmitted pkt_{sd}. Note that any non faulty intermediate node x that has received pkt_{sd} upstream will send back upstream either an ack_{sd}, a $failreport_x$ or a $failrport_y$, for some y, before its timeout. If s did not receive a valid $failreport_y$ for any y before its timeout, then $succ(s)$ must faulty and if s did receive a valid $failreport_x = [s, d, sn, seq, x, y]$ for some x, y before its timeout, then at least one of $\{x, y\}$ is faulty. In both cases s succeeds in tracing at least one faulty node. The full proof will be given in the journal version of this paper. □

In this tracing algorithm when there are no faults, a short ack is sent back. When faults do occur, a short $prob$ and $failreport$ are sent. In either case, a packet is confirmed successfully delivered, or a fault location is determined with only two digital signatures. This is the most efficient routing algorithm that will trace malicious behavior even when faulty nodes collude. It improves on the fault tracing algorithm in [1], which requires at least $\log(n)$ communication rounds and signatures to locate a malicious fault, and does not consider collusions.

Symmetric key. Symmetric key mechanisms such as hash chain and TESLA [25] can be used in our tracing protocol readily, which results in substantially faster signing, verification and tracing.

4.2 Tracing malicious behavior with AODV and DSR

Most of the routing algorithms can easily be extended to incorporate our tracing mechanism in the communication phase. For example, for distance vector based routings such as DSDV, AODV, and DSR , malicious faults will be traced by using the optimistic tracing algorithm for packet processing (the store-and-forward process). This can be done at the network layer, *i.e.*, after error checking at the data link layer (MAC).

5 Adaptive Multipath Routing

Multipath routing involves the establishment of multiple paths between source and destination pairs. These paths may be used for redundant communication to control malicious attacks. A major advantages in using multipaths is that, by exploiting redundancy we can guarantee service continuity, even when the adversary is active.

5.1 An Adaptive Multipath Routing algorithm

Finding routes with multiple paths in networks that do not have a fixed infrastructure is a challenge and in general requires a different approach to that used with fixed infrastructures. In this section we consider a multipath routing algorithm that combines in parallel a distributed version of Ford-Fulkerson Max Flow algorithm [6] (at the source) with a local network discovery algorithm (for nearby nodes) to find vertex-disjoint paths that link the source to the destination. When there are no malicious faults, a single route is used. Otherwise, the route is adaptively reconstructed to deal with the faults. Only the shortest route(s) is (are) are actually used, while the rest are kept alive.

The protocol is given in Figures 3 and 4. Figure 3 describes the actions of the source s. Initially s broadcasts a request req_s for neighbor lists. A hop-by-hop (on-the-fly) version of Ford-Fulkerson Max Flow algorithm[3] is used to construct a local graph $G^* = (V^*, E^*)$ with neighbor lists obtained from network nodes. G^* is a directed graph which is a vertex expanded version of the network graph G: each node x in G corresponds to two nodes x^+, x^- linked by (x^+, x^-) in G^*, and each link (x, y) of G corresponds to a link (x^-, y^+) in G^*, and conversely. Initially $G^* = \emptyset$. The source adds to G^* its neighbors and the links to them.The following variables are used:

- $flow$: a list of vertex-disjoint paths that link s to d in G^*; $value(flow)$: the number of paths in $flow$.

[3]The Ford-Fulkerson Max Flow algorithm is given for static networks. Here we consider an extension for mobile environments.

Source s

1. Set $G^* = \emptyset$, $flow = \emptyset$, $t = 1$, $radius = \Delta$.
2. Start using $flow$ for communication whenever $value(flow) \geq 1$.
3. $AddLinks(s, neighbors(s); flow, G^*)$.
4. While a connection to d has not terminated do
   (a) While $value(flow) < t$ do
      i. Set seq_s, ttl_s, $timeout_s$ and broadcast req_s.
      ii. For each valid rep_x received before $timeout_s$ do
         $AddLinks(x, neighbors(x); flow, G^*)$.
      iii. Set $radius = radius + \Delta$.
   (b) If $errorrate(path) > \epsilon_0$ for all $path \in flow$ then
      i. $t = t + 1$.

Figure 3: An adaptive multipath routing algorithm, I

- $req_s = [s, d, sn, seq_s, ttl_s]_s$: a request by s for neighbor lists consisting of identifiers for s, d, a session number sn, a sequence number seq_s for req_s, and the time-to-live ttl_s for req_s.
- $rep_x = [x, sn, seq_s, ttl_x, neighbor(x)]_x$: a report by x.
- $ctime_z$: the current time for node z.
- $radius$: an upperbound of the hop distance for req_s; Δ: an initial hop radius.
- $seq_s = ctime_s$; $ttl_s = ctime_s + radius \times \tau$; $timeout_s = ttl_s + radius \times \tau$.
- t: the number of disjoint paths of the multipath.
- ϵ_0: a threshold for the error rate of a non faulty path.
- $errorrate(path)$: the error rate of $path$.

Procedure $AddLinks(x, neighbors(x); G^*)$

1. $G^* = G^* + \{(x^+, x^-), (x^-, y^+), (y^+, y^-) \mid y \in neighbors(x)\}$.
2. Let $reverse(S) := \{(x, y) \mid (y, x) \in S\}$, for a set of links S of G.
3. For each path p from s^- to d^+ in G^* such that $p = (p - flow) + (p \cap reverse(flow))$, set $flow = flow + p - reverse(p)$.

Observe that each edge of G^* has capacity 1. Consequently $flow$ is a set of edge-disjoint paths in G^*. If $(s^-, x_1^+, x_1^-, \ldots, x_{n-1}^+, x_{n-1}^-, d^+)$ is a directed path in $flow$ then the corresponding path in G is $(s, x_1, \ldots, x_{n-1}, d)$ –provided all the reverse links (x_i^-, x_{i-1}^+) are also in G^*. It is not hard to see that if $\{(s^-, x_1^+, x_1^-, \ldots, x_{n-1}^+, x_{n-1}^-, d^+)\}$ is a set of edge-disjoint paths in G^* then the corresponding paths $\{(s, x_1, \ldots, x_{n-1}, d)\}$ in G are vertex-disjoint, and vice-versa.

Figure 4 describes the actions of the intermediate nodes and the destination. On receiving a request req_s each intermediate node x checks its validity and ttl_s. If these are in order, x sends a report rep_x to s with its neighbor list and forwards req_s. Similarly, when x receives a report rep_y from a y it checks its validity and ttl_y. If these are in order, x broadcasts rep_y.

Theorem 2 *The adaptive multipath routing algorithm tolerates any k-adversary, provided that the network graph is $(k+1)$-connected, $k \geq 1$.*

Intermediate node x and the destination

1. If a new valid req_s is received such that $ttl_s \leq ctime_x$ then
   (a) Set ttl_x and $timeout_x$.
   (b) Broadcast rep_x and req_s.
   (c) For each new valid rep_y received before $timeout_x$ do if $ttl_y \leq ctime_x$ then broadcast rep_y.

Figure 4: The adaptive multipath routing algorithm, II

Proof. (Sketch) If there are no faulty nodes then, when the source s requests local connectivity information from the nodes in radius Δ, each node in range will forward the request and reply with its list of neighbors. By $timeout_s$, s will have received a complete connectivity graph of the nodes that are no more than $radius$ hop counts from it. Observe that $radius$ increases adaptively, until s finds t disjoint paths from s to d, where $t \leq k+1$, and the graph is $(k+1)$-connected. Then, by the property of the Ford-Fulkerson algorithm, s will eventually succeed in finding t such paths. Note that since there are no malicious faults in this case, the value of t stays at 1.

Next consider the case when there are up to k malicious nodes. The faulty nodes may manipulate or fabricate packets but this will not affect the outcome of the algorithm because intermediate nodes always forward a new message before timeout, regardless of the actions or the states of their neighbors. Since we are assuming that the graph is $(k+1)$-connected, there must be a non faulty path between any pair of nodes. Consequently the request req_s of s will reach every intermediate node x in range, and conversely a report by any intermediate node x in range of req_s will always reach s.

In either case the route discovery always succeeds in finding routes. In the communication phase, the number of paths t needed increases adaptively until at least one good path is in the $flow$. Since the graph is $(k+1)$-connected, this process takes at most k steps, at which point $flow$ is assured to contains at least one non-faulty path. This adaptive approach avoids finding unnecessary paths when the adversary is partially active. The full proof will be given in the journal version of this paper. □

5.2 Discussion

The novelty of this route discovery algorithm is that it is resistant to malicious DoS attacks which are addressed adaptively. In particular, when there are no attacks a single route is used. With each malicious attack, the multipath is adaptively reconstructed to deal with the threat. Communication is activated as soon as a path becomes available, so there are no unnecessary delays.

In general when faults in a t-multipath occur beyond a certain acceptable threshold, the source s will use a $(t+1)$-multipath. Since the new set of paths is already constructed in the background, the delay caused by faults is minimized. Most of the time, there should be no delay. Furthermore, in our algorithm, the set of vertex-disjoint paths of the multipath is constructed incrementally, so that even when delays are unavoidable, they are minimal.

For efficiency, each node on a path only needs to know its upstream and downstream neighbor. So the path information needs to be sent to intermediate nodes only at the beginning. When changes are made to the multipath, the source needs only send the changes to all nodes on the new paths. The nodes will discard unused information after a period of inactivity.

Observe that having local information available centrally is more effective than having it distributed. In particular, the procedure used in the adaptive routing algorithm by the source allows more vertex-disjoint paths to be found than by the distributed process used in most other multipath routing protocols (because all the routing information is available locally). As a consequence fewer communication rounds may be needed when faults occur.

Finally, observe that we can combine our adaptive multipath routing algorithm with the Dynamic Source Routing algorithm [16] to get an adaptive multipath DSR algorithm. Similarly, we may combine the adaptive multipath routing algorithm with the tracing mechanism in Section 4.1 to get an adaptive routing algorithm that will trace malicious behavior.

References

[1] B. Awerbuch, D. Holmer, C. Nita-Rotaru and H. Rubens, *An On-Demand Secure Routing Protocol Resilient to Byzantine Failures*, ACM Workshop on Wireless Security – WiSe'02 2002.

[2] D. Beaver, *Foundations of secure interactive computing*, Proc. CRYPTO '91, Springer Verlag LNCS, vol. 576, pp. 377-391, 1991.

[3] E.M. Belding-Royer and C.-K. Toh, *A review of current routing protocols for ad-hoc mobile wireless networks*, IEEE Personal Communications Magazine, pp. 46-55, 1991.

[4] M. Burmester, Tri van Le and A. Yasinsac. *Adaptive gossip protocols: managing security and redundancy in dense ad hoc networks.* Journal of Ad hoc Networks, Elsevier, 2006.

[5] J. R. Douceur, *The Sybil attack*, Proc. 1st International Workshop on Peer-to-Peer Systems – IPTPS '02, 2002.

[6] L.R. Ford and D.R. Fulkerson, Flows in Networks. Princeton University Press, *Princeton*, NJ, 1962.

[7] D. Cavin, Y. Sasson and A. Schiper, *On the Accuracy of MANET Simulators*. Proc. of the 2nd ACM international workshop on Principles of mobile computing, Toulouse, France, pp.38-43, 2002.

[8] M. Felegyhazi, L. Buttyan and J.-P. Hubaux, *Equilibrium analysis of packet forwarding strategies in wireless ad hoc networks –the static case*. Lecture Notes in Computer Science #2775, Springer-Verlag, pp. 776–789, 2003.

[9] O. Goldreich, S. Micali, and A. Wigderson. *How to play any mental game.* Proc. of the 19th ACM conference on Theory of Computing, ACM Press, pp. 218–229, 1987.

[10] Z.J. Haas, J.Y. Halpern and L. Li. *Gossip-based ad hoc routing*. Proc. INFOCOM'02, pp. 1707-1716, 2002.

[11] M. Hirt and U. Maurer, Player Simulation and General Adversary Structures in Perfect Multiparty Computation, *Journal of Cryptology*, Vol 13 No 1, pp. 31-60, 2000.

[12] Y-C Hu, D.B. Johnson and A. Perrig. *Ariadne: A Secure On-Demand Routing protocol for Ad Hoc Networks.* ACM Mobicom 2002.

[13] Y-C Hu, D.B. Johnson and A. Perrig. *SEAD: Secure Efficient Distance Vector Routing for Mobile Wireless Ad Hoc Networks.* Proc. 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), IEEE, Calicoon, NY, 2002.

[14] Y-C. Hu, A. Perrig and D.B. Johnson. *Rushing attacks and defense in wireless ad hoc network routing protocols* – WiSe2003, pp. 30-40, 2003.

[15] A. Jardosh, E. M. Belding-Royer, K. C. Almeroth and S. Suri, *Towards realistic mobility models for mobile ad hoc networks*, Proc. 9th Annual International Conference on Mobile Computing and Networking, pp. 217-229, 2003.

[16] D.B. Johnson and D.A. Maltz, *Dynamic Source Routing in Ad-Hoc Wireless Networks*, ed. T. Imielinski and H. Korth, Mobile Computing, Kluwer Academic Publisher, pp. 152-181, 1996.

[17] G. Koh, D. Oh and H. Woo, *A graph-based approach to compute multiple paths in mobile ad hoc networks*, Lecture Notes in Computer Science #2713, Springer-Verlag, pp. 323–331, 2003.

Proc. ACM/IEEE MOBICOM '98 (1998).

[18] G. Lin, G. Noubir and R. Rajaraman, *Mobility Models for Ad hoc Network Simulation.* Proc. IEEE INFOCOM, 2004.

[19] S. Micali, P. Rogaway, *Secure Computation*, Crypto '91, LNCS 576, pp. 392-404, 1991.

[20] P. Papadimitratos and Z.H. Haas. *Secure Routing for Mobile Ad hoc Networks*. Mobile Computing and Communications Review, Vol 6, No 4, 2002.

[21] C.E. Perkins and P.Bhagwat, *Highly Dynamic Destination-Sequenced Distance-Vector Routing for Mobile Computers*, Computer Communications Review, pp. 224-244, 1994.

[22] C.E. Perkins and E.M. Royer, *Ad hoc on-demand distance vector routing*, IEEE Workshop on Mobile Computing Systems and Applications, pp. 90-100, 1999.

[23] N. Salem, L. Buttyan, J. Hubaux and M. Jakobsson, *A charging and rewarding scheme for packet forwarding in multi-hop cellular networks*. In Mobihoc 2003, pp. 13-24, 2003.

[24] A.J. Menezes, P.C. van Oorschot and S.A. Vanscott, *Handbook of Applied Cryptography*, CRC Press, 1996.

[25] A. Perrig and R. Canetti and D. Tygar and D. Song, *The TESLA Broadcast Authentication Protocol*, Cryptobytes, Volume 5, No. 2, pp. 2–13, RSA Laboratories, 2002.

[26] A. Tsirigos and Z.J. Haas Analysis of multipath routing, part 1: The effect on the packet delivery ratio, *IEEE Transactions on Wireless Communications*, Vol. 3, No. 1, pp. 138-146, 2004.

[27] Jungkeun Yoon, Mingyan Liu, Brian Noble, *Random Waypoint Considered Harmful*. Proc. IEEE INFOCOM, 2003.

[28] M.G. Zapata. *Secure Ad hoc On-Demand Vector (SAODV) Routing*. IETF Internet Draft. draft-guerrero-manet-saodv-00.txt. Aug 2001 (work in progress).