ODSBR: An On-Demand Secure Byzantine Routing Protocol

Baruch Awerbuch [*]   Reza Curtmola [*]   David Holmer [*]   Cristina Nita-Rotaru [†]
Herbert Rubens [*]

Abstract

A common technique used by routing protocols for ad hoc wireless networks is to establish the routing paths on-demand, as opposed to continually maintaining a complete routing table. Since in an ad hoc network nodes not in direct range communicate via intermediate nodes, a significant concern is the ability to route in the presence of Byzantine failures which include nodes that drop, fabricate, modify, or mis-route packets in an attempt to disrupt the routing service.

We propose the first on-demand routing protocol for ad hoc wireless networks that provides resilience to Byzantine failures caused by individual or colluding nodes. The protocol relies on an adaptive probing technique that detects a malicious link after $\log n$ faults have occurred, where n is the length of the path. Problematic links are avoided by using a weight-based mechanism that multiplicatively increases their weights and by using an on-demand route discovery protocol that finds a least weight path to the destination. Our protocol bounds the amount of damage that an attacker or a group of colluding attackers can cause to the network.

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

Terms: Algorithms, Design, Reliability, Security, Theory

Keywords: ad hoc wireless networks, on-demand routing, security, Byzantine failures

1   Introduction

Ad hoc wireless networks are self-organizing multi-hop wireless networks where all the nodes take part in the process of forwarding packets, facilitating communication between nodes not in direct range. Ad hoc networks can easily be deployed since they do not require any fixed infrastructure, such as base stations or routers. Thus, they are highly applicable to communication in regions with no fixed infrastructure such as emergency deployments, natural disasters, military battle fields, 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. Ad hoc wireless routing protocols must converge quickly and use battery power efficiently, to overcome the challenges presented by high mobility and constrained power resources. Traditional proactive routing protocols using periodic updates (link-state and distance vectors [1]) are less suitable for ad hoc wireless networks because they constantly consume power and bandwidth 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

---

[*] Department of Computer Science, Johns Hopkins University, 3400 North Charles St. Baltimore, MD 21218 USA. E-mail: {baruch, crix, dholmer, herb}@cs.jhu.edu .

[†] Department of Computer Science, Purdue University, 250 N. University Street, West Lafayette, IN 47907 USA. E-mail: crisn@cs.purdue.edu .

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.

A major requirement for routing protocols is the ability to operate in adversarial environments. For example, a malicious node may advertise false routing information, try to redirect routes, perform a denial of service attack by engaging a node in resource consuming activities, or simply drop packets. An attack specific to wireless networks is a wormhole when one or more adversaries convince a pair of legitimate nodes that a link exists between them by propagating packets sent by one node to the other and thus disrupting the routing service. We note that due to their cooperative nature and the broadcast medium, ad hoc wireless networks are more vulnerable to attacks in practice [4].

Prior work in secure routing considered that the participants in the network should be trusted (with very few exceptions, such as [5, 6, 7]). We argue that particularly for an ad hoc wireless network, it is not always the case 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. In addition, mobile wireless devices can be compromised more easily (e.g. lack of physical security), so complete trust should not be assumed.

Our contribution. The goal of this work is to provide routing survivability under an adversarial model where any intermediate node or group of colluding nodes can perform Byzantine attacks such as creating routing loops, misrouting packets on non-optimal paths, or selectively dropping packets; only the source and the destination nodes are assumed to be trusted. To our knowledge, there is no on-demand wireless routing protocol addressing Byzantine failures, particularly in a model where attackers can collude.

A Byzantine fault occurring along a path may be attributed to a specific node using expensive and complex Byzantine agreement; however, this is provably impossible under certain circumstances, for example when a majority of the nodes are malicious. We circumvent this obstacle by avoiding the assignment of "guilt" to individual nodes. Instead, whenever the endpoints of a link disagree, we deduce that at least one of them is faulty, therefore the link is considered faulty and should be avoided. Our method ensures that 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.

This work is an enhanced version of our previous work [8]. In this paper, we present a route discovery phase that is based on aggregate signatures [9], as opposed to chained digital signatures. This approach provides stronger security guarantees and improved convergence times. The fault detection phase has been redesigned so that onion encryption is no longer required, thus reducing computational cost and packet size overhead. We also describe an efficient scheme for securely boot strapping shared keys from an existing public key infrastructure.

More specifically our contributions are:

- We provide a secure robust on-demand routing protocol which is resilient to strong adversarial attacks including those performed by colluding Byzantine attackers.

- Using an adaptive probing technique, the protocol identifies a faulty link after $\log n$ faults have occurred, where n is the length of the path.

- We provide an upper bound on the damage an attacker or group of colluding attackers can cause to the network. Our link monitoring mechanism limits the amount of damage, while never completely disconnecting nodes from the network.

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

2   Network and Security Model

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 required by most wireless MAC protocols, including 802.11 [10], to operate correctly. We focus on providing a secure routing protocol, which specifically addresses threats to the ISO/OSI network layer. We do not address attacks against lower layers. 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 [11].

Security Model and Considered Attacks   In this work we consider only the source and 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 is expected to perform the protocol correctly, but may exhibit Byzantine behavior. The goal of our protocol is to detect Byzantine behavior and avoid it. We assume that an intermediate node can exhibit such behavior either alone or in collusion with other nodes.

We define *Byzantine behavior* as any action by an authenticated node that results in disruption or degradation of the routing service. Examples of such attacks include intercepting, modifying, or fabricating packets, creating routing loops, dropping packets selectively (often referred to as a *black hole*), artificially delaying packets, or making a path look either longer or shorter than it is.

Our protocol is also resilient to several forms of strong non-Byzantine attacks, including wormholes and flood blocking attacks. These types of attacks are non-Byzantine because they can be executed by un-authenticated adversarial nodes, however they are considered strong attacks and are not handled by the majority of existing secure routing protocols.

Wormhole attacks are when one or more adversaries convince a pair of legitimate nodes that a link exists between them by propagating packets sent by one node to the other. Our protocol addresses this wormhole attack by treating the wormhole as a single link that will be avoided if it exhibits losses or excessive delay, but it does not prevent wormhole formation.

A flood blocking attack is a network level denial of service attack that exploits the flood duplicate suppression technique used by many routing protocols. An adversary may corrupt the flooded packet, and if the corrupt version reaches a node before a valid version, the valid flood will be blocked by the flood duplicate suppression. This may cause a continual inability to establish a valid route, even when using end-to-end or path authentication techniques. Our protocol handles this attack by performing hop-by-hop verification at every step of the flood, thus preventing any invalid information from propagating. Also, our protocol is not vulnerable to adversarial manipulation of flood timing since it forwards the flood with the lowest metric rather than the first flood received.

Whenever possible, our protocol uses efficient cryptographic primitives. In order to accomplish this, we employ on-demand shared keys established as described in Section 3.2. However, a public-key infrastructure is still required for other operations such as route discovery and shared key establishment. This infrastructure can be either completely distributed (as described in [12]), 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.
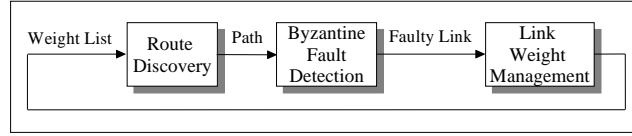
Figure 1. Secure Routing Protocol Components

3   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 regular 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 mechanisms, and as input a list with the weights of faulty links, this component outputs the full least weight path from the source to the destination. It is invoked every time a new path needs to be discovered.

- *Byzantine fault detection.* This component discovers faulty links on the path from the source to the destination using as input the full path and outputting a faulty link. The key is an adaptive probing technique that identifies a faulty link after $\log n$ faults have occurred, where n is the length of the path. Cryptographic mechanisms and sequence numbers are used to protect the detection protocol from adversaries.

- *Link weight management.* One goal of our protocol is fault avoidance. This is achieved by the the route discovery phase based on weights associated with each link. The link weight management component of the protocol maintains a weight list for links discovered by the fault detection algorithm and uses a multiplicative increase scheme to penalize links. In addition, our protocol uses a rehabilitation mechanism that limits the amount of damage an attacker or group of attackers can cause, without disconnecting nodes completely.

3.1   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 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.

Our route discovery protocol uses link weights to avoid faults. A weight list is provided by the link weight management phase (Section 3.3). 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. At the completion of the route discovery protocol, the source is provided with a complete path to the destination. Many routing protocols use route caching by intermediate nodes as an optimization; we do not consider it in this work because of the security implications, and defer it for future work.

In order to protect the route discovery phase from external or Byzantine attackers, several protection mechanisms are used. During the initial route request flood, a digital signature is used to authenticate the source. This is required to prevent unauthorized nodes from initiating resource consuming route requests, and to limit excessive route requests from a Byzantine node. 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.

In our protocol, the actual path discovery occurs during the route response flood, as opposed to during the route request flood as in other on-demand protocols. This reduces the cost of route requests to unreachable destinations and allows the destination's learned weight list to be used. In order to protect the accumulated path, it is essential that the protocol: allow only authenticated nodes to become part of the path, prevent any adversary from being able to modify the accumulated path, as well as allow every node to be able to verify the entire contents of the packet so far to prevent a flood blocking attack. Our protocol uses a protection mechanism based on the aggregate signature scheme introduced by Boneh et al. in [9] to provide all three of the required properties.

Aggregate Signatures: Given n users that sign n messages, an aggregate signature allows the composition of these digital signatures into a single short signature. An aggregate signature only verifies correctly if the verifier is provided with the exact contents signed by, and the exact identity of, every party that contributed to the aggregate. It is important to note that the aggregation can be performed incrementally; in our scenario, if a node receives signature σ_{12} (obtained by aggregating σ_1 and σ_2), then it can further aggregate it with σ_3 to obtain σ_{123}. Furthermore, given an aggregate signature, it is cryptographically difficult to remove the contribution of one of the signers without knowledge of their private key.

In addition, aggregate signatures have the advantage that their size does not increase with the number of signers, so the protocol overhead is minimized. However, the disadvantage of aggregate signatures is their high computational cost. In spite of significant speed improvements [13], the verification of an aggregate signature is still more expensive than verifying an equivalent number of RSA signatures.

We note that while the chained digital signature scheme used in our prior work requires less computation, it does not provide the same level of security because it does not fully protect the path from adversarial modification. An adversary may strip any number of hops from the end of the accumulated path, thus creating a non-functional "virtual" link. The aggregate scheme is immune to this form of attack because contributions to the aggregate signature cannot be easily removed.

The following five steps comprise the route discovery protocol:

I. Request Initiation.   The source creates and signs a request that includes the destination, the source, a sequence number, and a list of detected malicious links and their weights. 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 verifies the source signature on the request and checks its list of recently seen requests for a matching request (one with the 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. 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, and creates a response that contains the source, the destination, a response sequence number and the combined link weight list (both the source and local weights are merged). The destination then signs the entire response using an aggregate signature and broadcasts it.
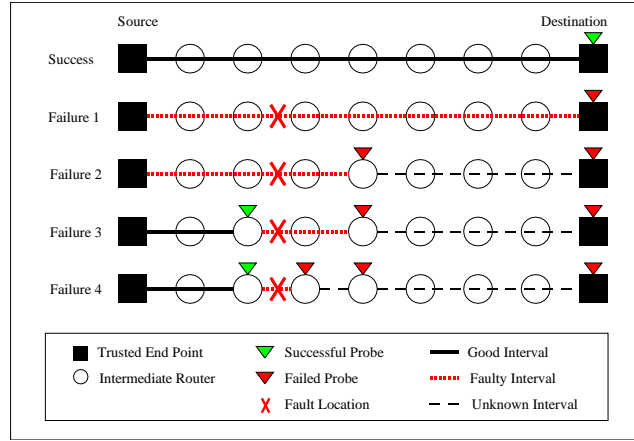
Figure 2. Byzantine Fault Detection: an Example

IV. Response Propagation. When receiving a response, the node verifies the aggregate signature. This is accomplished by using the accumulated path list including the destination as the set of signers, and by treating the response packet contents up to and including the signer's id as the data signed by that signer. The total weight of the path is computed by summing the weight of all the links on the specified path up to this node. If the entire packet is verified and the total weight is less than any previously forwarded matching response (same source, destination, and sequence number), the node appends its identifier to the end of the list, adds its own signature of the modified response in order to generate the new aggregate signature, and broadcasts the modified response. This response propagation functions similarly to the single source Bellman-Ford algorithm.

V. Response Receipt. When the source receives a response, it performs the same 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.

3.2 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 was lost. This definition of loss includes both malicious and non-malicious causes. For example 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 just a drop of the packets by intermediate nodes.

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. A *fault* is defined as a loss rate greater than or equal to the threshold. The value of the threshold also specifies the amount of loss that an adversary can create without being detected. Hence, its value 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.

Our protocol is designed to have minimum overhead 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 by requiring a dynamic set of intermediate nodes, in addition to the destination, to send acks to the source. Moreover, the fault detection protocol is built to primarily use symmetric key cryptography, so costly asymmetric operations are avoided on a per packet basis.

seq,data,ID_1,ID_2,ID_3,...,ID_p,$HMAC_{dest}$,$HMAC_p$,...,$HMAC_3$,$HMAC_2$,$HMAC_1$ where $HMAC_i$ is computed with K $_{source,ID_i}$ and p is the number of probes

Figure 3. Probe Specification

Normal topology changes occur frequently in ad hoc wireless networks. Although our detection protocol locates faulty links caused by these changes, an optimized mechanism for detecting them would decrease the overhead and detection time. Any of the mechanisms used by the route maintenance of the DSR protocol [3], (i.e. MAC layer notifications), can be used as an optimized topology change detector. When a node receives notifications from such a detector, it reacts by creating a route error message containing the identifiers of the broken link and of the node that reports the error. The signed message is then propagated along the path back to the source. Upon receipt of an authenticated route error message, the source passes the faulty link to the link weight management component. 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 Mechanism.   Our fault detection protocol requires the destination to return an ack to the source for every received data packet. The source keeps track of the number of recent losses (acks not received over a window of recently sent packets). If the losses violate 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 2.

The source controls the search by specifying a list of intermediate nodes on data packets that also must send acks. We refer to the set of nodes required to send acks as probed nodes, or for short *probes*. Since the list of probes is specified on legitimate traffic, an adversary is unable to drop traffic without also dropping the list of probes 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 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 to the link weight management component (see Figure 1). The path sub-division process is a binary search that proceeds one step for each fault detected. This results in the detection of a faulty link after $\log n$ faults have occurred, where n is the length of the path.

The probes are specified by listing the identifiers of the probed nodes in path order on each packet. This list is protected from tampering by appending an HMAC [14] (computed with the shared key between the source and that node) of the entire packet so far for every probed node in the reverse order they are specified on the packet (see Figure 3). A node can detect if it is required to send an ack by checking the list for its identifier. If the node finds its identifier, it verifies the last HMAC and removes it from the packet. A node discards packets that do not have the correct number of remaining HMACs, and a probe discards packets if the last HMAC does not verify. The reverse ordered HMACs prevent the adversary from incriminating other links by successfully tampering with the node list (i.e. removing specific nodes). If an adversary attempts to modify the list, it will incriminate one of its own links.

Acknowledgment Specification.   For each successfully received data packet, the destination generates an ack packet containing the sequence number of the data packet and an HMAC for authentication. Each probe appends its own HMAC to the ack and forwards it along the reverse path towards the source. Timeouts are set at every probe so that if the ack is not received, the node gives up waiting and generates its own ack packet. In order for the

7

protocol to locate the fault, only the node immediately preceding the fault location should timeout and generate an ack. To accomplish this, a staggered timeout scheme is utilized. A simple technique for implementing the staggered timeout scheme is to use a fixed parameter that represents an upper bound on the amount of time it takes for a packet to traverse a single link. A node may then compute its timeout duration by multiplying this parameter by the number of hops needed to traverse the remainder of the path to the destination and return back to the current position along the path.

When the source receives the ack packet, it attempts to verify the accumulated HMACs starting from the end of the packet. The first HMAC, if any, that does not verify, defines the end of the interval on which a loss is registered, i.e. the protocol registers a loss on the interval between the last valid HMAC and the first encountered invalid HMAC. If the source times out the ack, a loss is registered on the interval between the source and the first probe. The result of this ack creation, timeout, and verification technique, is that if an adversary drops, modifies, or delays either an ack or data packet, then a loss will be registered on the interval containing the affected link.

Interval and Probe Management. Let ρ be the acceptable threshold loss rate. By using the above probe and acknowledgment specifications 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 ρ, 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 ρ and the number of lost packets. The goal is to bound the aggregate loss rate. Each interval has an associated counter C that specifies its lifetime. Initially, there is one interval with a counter of zero (i.e. there are no losses). 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 $\mu/\rho + C$, 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. This results in a loss rate bounded to ρ. If the adversary attempts to create a higher loss rate, the algorithm will be able to identify the faulty link.

Shared Key Establishment. The Byzantine Fault Detection phase makes extensive use of pairwise symmetric keys shared between the source and each node along the path. Pre-distributing and refreshing shared keys would be impractical for network maintainers. We propose a technique for on-demand creation of these keys using the assumed public key infrastructure. When the source needs to probe a node with which it does not already share a key, it generates a new random key for that node, and encrypts the new key with that node's public key, so only that node will be able to recover the key. The encrypted key is embedded it in the probe list on outgoing data packets. Digital signatures are used to authenticate the packets before the shared key is established. Once the shared key is established acks sent by the node can be authenticated using an HMAC.

In order to ensure reliable delivery of the shared key to the intended node even in the case where there are active adversaries disrupting the path, the source continues to attach the encrypted key for every outgoing data packet, until a verifiable ack (that uses the shared key) is received from the node. Note that since the shared key establishment is seamlessly integrated with the adaptive probing, multiple keys can be established simultaneously even as the protocol dynamically changes the set of probed nodes.

It may be desirable to generate the key shared by the source and destination using an authenticated contributory key exchange instead of the above technique. This is because the source destination key will likely be used to encrypt data, and thus perfect forward secrecy [1] would be desirable. This property is not necessary for interme-

---

[1] Informally, perfect forward secrecy [15] demands that the compromise of long-term keys should not lead to the compromise of any previously used session keys

diate nodes as their shared keys are used only for authentication and integrity checks, not to protect data. An authenticated Diffie-Hellman [15] key exchange can either be piggy-backed on the route request and response floods, or conducted immediately after the path has been established.

3.3 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 component of the protocol. The link management scheme sets the link weights using the history of faults that have been detected on links. A faulty link is penalized by doubling its weight.

The technique we use for resetting the link weights is similar to the one we use for retiring probes (see Section 3.2). 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 and ρ is the threshold loss rate, then the link's counter is increased by μ/ρ. 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.

4 Analysis

In this section we provide an upper bound on the number of packets lost because of adversarial behavior, namely behavior that reduces the packet transmission success rate below some threshold.

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 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 q^-/μ, so $q^-/\mu - \sum_{e \in \tilde{E}} j_e < 0$. Also, the number of rehabilitations is at most $q^+/\mu/\rho$, so $\sum_{e \in \tilde{E}} a_e - q^+/\mu/\rho < 0$, where μ is the number of lost packets that exposes a link as faulty. Thus

$$\frac{q^-}{\mu} - \frac{q^+}{\mu/\rho} \leq \sum_{e \in \tilde{E}} (j_e - a_e) \tag{3}$$

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

$$\sum_{e \in \tilde{E}} (j_e - a_e) = \sum_{e \in \tilde{E}} \log w_e \tag{4}$$

By combining Eq. (3) and (4), we obtain

$$q^- - \rho \cdot q^+ \leq \mu \sum_{e \in \tilde{E}} \log w_e \leq \mu \cdot kn \cdot \log n \tag{5}$$

and since $\mu = b \log n$, where b is the number of lost packets per window, Eq. (5) becomes

$$q^- - \rho \cdot q^+ \leq b \cdot kn \cdot \log^2 n \qquad (6)$$

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 are no adversarial nodes Eq. (6) becomes the ideal case where $q^- - \rho \cdot q^+ \leq 0$.

5   Related Work

A key component of providing security services in an ad hoc wireless networks is an effective public key infrastructure. Research results in this direction are as follows. Hubaux et al. [12] proposed a completely decentralized public-key distribution system similar to PGP [16]. Zhou and Haas [17] explored threshold cryptography methods in a wireless environment. Brown et al. [18] showed how PGP, enhanced by employing elliptic curve cryptography, is a viable option for wireless constrained devices.

As noted in [19], source authentication is more of a concern in routing than confidentiality. Papadimitratos and Haas showed in [20] 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 [14] primitive which relies on the existence of security associations between sources and destinations. Other significant works include SEAD [21] and Ariadne [4] that provide efficient secure solutions for the DSDV [22] and DSR [3] routing protocols. SEAD uses one-way hash chains to provide authentication, while Ariadne uses Tesla [23] source authentication technique to achieve similar security goals.

In [24] the authors focus on an analogous problem, providing end-to-end authentication for two well-known on-demand protocols: AODV [2] and DSR [3], by using a strong, but expensive, authentication means: digital signatures. They also provide an expensive protocol that guarantees minimum path selection using an onion [25] like technique, where digital signatures and public cryptography encryption/decryption are performed and accumulated at each hop.

A strong attack specific to wireless networks, referred to as a *wormhole* [4], is where two attackers establish a path and tunnel packets from one to another. These packets can arrive faster than the normal route request flood, which can result in establishing non-optimal, adversarial controlled routing paths. Recent work by Hu et al.[26] proposes a solution to this by limiting the distance a packet can travel.

Marti et al. [6] address a problem similar to the one we consider, survivability of the routing service when nodes selectively drop packets, by taking advantage of the wireless cards promiscuous mode. Trusted nodes are monitoring their neighbors, while links with an unreliable history are avoided in order to achieve robustness. Although the idea of using the promiscuous mode is interesting, the solution does not work well in multi-rate wireless networks where a modulation scheme is adaptively selected based on the current channel conditions when a node transmits.

Relevant work has been conducted also in the wired network community. Many researchers focused on securing classes of routing protocols such as link-state [19, 27, 28, 29] and distance-vector [30], or well-known protocols such as OSPF [31] and BGP [32]. The problem of source authentication for routing protocols was explored using digital signatures [31] or symmetric cryptography based methods: hash chains [19], chains of one-time signatures [28] or HMAC [29].

Perlman [5] 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 [33, 34]. The approach in [33] 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 [34]. 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 achieved 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.

6 Conclusions and Future Work

We presented a secure on-demand routing protocol resilient to Byzantine failures caused by an adversary or group of colluding adversaries. A key component of our scheme is an adaptive probing technique that detects malicious links after $\log n$ faults have 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.

The protocol we presented can be enhanced in many aspects. One example is to use route caching at intermediate nodes. We do not consider route caching optimizations in this work, but we intend to explore it in the future. Next, we plan to evaluate the overhead of our protocol with respect to existing protocols, in normal, non-faulty conditions as well as in adversarial environments.

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.

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

[6] 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.

[7] P. Papadimitratos and Z. Haas, "Secure data transmission in mobile ad hoc networks," in 2nd *ACM Workshop on Wireless Security (WiSe)*, September 2003.

[8] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *ACM Workshop on Wireless Security (WiSe)*, September 2002.

[9] D. Boneh, C. Gentry, H. Shacham, and B. Lynn, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proceedings of Advances in Cryptology - Eurocrypt'03*, LNCS, 2003.

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

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

[12] 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.

[13] P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," in *Proceedings of Advances in Cryptology - CRYPTO 2002*, LNCS, 2002.

[14] *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.

[15] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

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

[17] L. Zhou and Z. Haas, "Securing ad hoc networks," *IEEE Network Magazine*, vol. 13, no. 6, 1999.

[18] 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.

[19] 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.

[20] 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.

[21] 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*, June 2002.

[22] 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.

[23] 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.

[24] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. Belding-Royer, "A secure routing protocol for ad hoc networks," in *10th IEEE International Conference on Network Protocols (ICNP'02)*, November 2002.

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

[26] Y.-C. Hu, A. Perrig, and D. B. Johnson, "Packet leashes: A defense against wormhole attacks in wireless ad hoc networks," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, April 2003.

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

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

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

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

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

[32] 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.

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

[34] 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.