Routing amid Colluding Attackers

Jakob Eriksson
MIT CSAIL
jakob@csail.mit.edu

Michalis Faloutsos, Srikanth V. Krishnamurthy
University of California, Riverside
michalis, krish@cs.ucr.edu

***Abstract*— We propose the first practical solution to the long-standing problem of secure wireless routing in the presence of colluding attackers. Our secure routing protocol, Sprout[1], continuously tries new routes to the destination. Routes are probabilistically generated, with complete disregard for performance metrics. This makes Sprout uniquely resilient to attack: it cannot be tempted by shortcuts. In order to avoid compromised routes, and to ensure good overall performance, the quality of each active route is monitored by means of signed end-to-end acknowledgments. The amount of traffic sent on each route is adjusted accordingly. Sprout effectively mitigates the vast majority of known routing layer attacks, even when under assault from a large number of colluding attackers. Experiments on our 31-node testbed demonstrates the real-world performance of Sprout in terms of packet delivery ratio, round-trip times and TCP throughput. Our security analysis and simulation results show that Sprout is able to quickly find working paths in networks of hundreds of nodes and dozens or more attackers. For example, in a network of 200 nodes and an astounding 64 attackers, Sprout, on average, found a successful route within less than 10 attempts. Yet, in benign settings, Sprout provides TCP throughput within 15% of the shortest path throughput. Overall, Sprout consistently delivers high, reliable performance in benign as well as hostile environments.[2]**

I. Introduction

Routing protocols are subject to a wide variety of attacks, many of which can be highly disruptive. Many sophisticated attacks are "insider attacks", in which the attacker has access to legitimate nodes or certain cryptographic credentials. Attacks by independent insiders have been addressed in the literature, including [2]–[7]. However, much of the previous work focuses on providing a secure environment for "insider" nodes with respect to attacking "outsiders". Current secure routing protocols rarely address attacks by multiple colluding insiders. The problem becomes more pronounced in open networks, where nodes are considered legitimate members by default. Previous work on probabilistic routing focuses on selfish routing, and does not directly address routing security.

We propose Secure Probabilistic Routing (Sprout), a practical solution to the long-standing problem of secure wireless routing in the presence of *multiple colluding insider attackers*. Sprout effectively mitigates the vast majority of the known routing layer attacks, and provides good performance in benign conditions. Sprout is a source-routed, link-state, multi-path routing protocol. In contrast with previous work, Sprout generates routes probabilistically, focusing in the first stage on diversity, rather than predicted performance. This makes it more resilient than previously proposed routing algorithms, to a wide variety of attacks. The obvious drawback of this approach is that many of the generated paths are of poor quality, and may include attackers. To address this, a performance based path selection algorithm is used as a second stage, which assigns a probability to each generated route depending on its measured reliability and end-to-end delay. Reliable routes with short round-trip times carry the majority of packets, while a fraction of packets are sent along other routes, to maintain diversity. With every new route sampled, the probability of finding a good one increases rapidly.

The primary contributions of this paper are as follows:

- A secure link-state dissemination protocol that minimizes the types of fake links available to colluding attackers.
- A probabilistic route generation algorithm that quickly finds routes through massively polluted link-state graphs.
- A probabilistic route selection algorithm that effectively balances security and performance, and
- A comprehensive evaluation, through analysis, simulation and implementation, of the above.

We have implemented Sprout in Linux, and deployed it on our 31-node indoor wireless testbed. Our experiments demonstrates the real-world performance of Sprout in terms of packet delivery ratio, round-trip times and TCP throughput. Sprout effectively mitigates a wide range of attacks, and delivers consistently high, reliable performance in hostile environments. For example, in a network of 200 nodes and an astounding 64 attackers, Sprout, on average, found a successful route within less than 10 attempts. Similarly, in a massive sybil attack with almost 3 times as many attacker identities as legitimate identities (576 vs. 200), Sprout was on average able to find a successful path after 35 attempts. While the focus of this work is on security, rather than performance, our evaluation shows that even in benign conditions, Sprout performance is within 15% of that of shortest path routing.

The rest of the paper is structured as follows. Sec II provides an overview of prior related work In Sec. III, we describe the Sprout protocol in detail. Sec. IV analyses Sprout's resilience to routing attacks. Sec. V assesses Sprout's high-level performance in large networks, facing massive, sophisticated attacks. Sec. VI describes our implementation of Sprout, and its performance on our experimental testbed.

[1]The name "Sprout" has been used before. In [1], the SPROUT DHT social network routing algorithm was proposed. We apologize for any confusion.

[2]Prepared partially through collaborative participation in the Communications and Networks Consortium sponsored by the U. S. Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement DAAD19-01-2-0011, and partially with support from the U.S. Army Research Office under the Multi-University Research Initiative (MURI) grant W911NF-07-1-0318. The U. S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

II. Related Work

Previous work on secure routing [2]–[6], [8] has addressed independent attackers. However, attacker collusion is frequently, and imprudently, disregarded. In addition to regular attacks, colluding attackers can a) pose as each other, creating the appearance of links that do not exist, b) vouch for each other, making attacking nodes appear legitimate, and c) collude to create evidence against legitimate nodes, in a $blackmail$ attack, making them appear to be malicious or defective.

In general, previously proposed techniques are insufficient for dealing with colluding attackers. A commonly used secure routing technique is node-disjoint k-path routing. This technique cannot provide routing security in any network that is not k-connected. The problem is dramatically compounded by colluding attackers; in the absence of specific security measures, colluding attackers can frequently manipulate the routing metric to ensure that the shortest $k-1$ disjoint paths all contain attacker nodes. Sprout does not require node- or edge-disjoint paths.

Avoiding links or nodes after they display malfunctioning or malicious behavior is another common secure routing technique [5]. The difficulty lies in accurately determining what node is malfunctioning; attackers may be able to implicate other nodes, enabling a powerful denial-of-service attack. ODSBR [7] tracks down faulty links using signed acknowledgments from intermediate relay nodes. ODSBR is highly susceptible to the *sybil* and *wormhole* attacks, where colluding attackers may create a large number of fictitious links, all of which must be identified as bad before the protocol succeeds. Sprout bases routing decisions on $route$ performance rather than individual links or nodes.

Other related work includes [9], which assumes an accurate link-state graph, and the probabilistic techniques proposed in [10], [11]. These do not address routing security. In SMT [12], the authors assume the existence of multiple paths of varying reliability, and design an end-to-end secure message transmission protocol that exploits these paths to ensure maximum reliability and throughput. The SMT approach for end-to-end reliability is well suited for use together with Sprout.

In [13], Perlman describes a secure flooding protocol for wireline networks, and a link-state routing protocol which are highly resilient to byzantine and denial-of-service attacks. This work shares some aspects with Perlman's early efforts, and we use the secure flooding protocol as described in that work.

III. Secure Probabilistic Routing (Sprout)

Sprout is a **source-routed**, **link-state**, **multi-path** routing protocol with a probabilistic twist. Routing is done in two stages: route generation, and route selection. In the route generation stage, a large number of routes is probabilistically generated, independent of any routing metric. This initial disregard for performance, resulting in a highly diverse set of routes, is fundamental to the security of the protocol In the route selection stage, the performance of each active route is monitored by means of signed end-to-end acknowledgments. The reliability and round-trip time of an active route determines the fraction of packets sent over it.

A. Scenario and Security Model

We consider a multi-hop wireless network, and a very strong attacker model. Attackers control multiple nodes, and attacker nodes are potentially connected through high-speed out-of-band communication links. This is a reasonable attacker model: attackers can simply use existing alternative means of communication. Furthermore, attackers are able to "spoof" MAC addresses and/or any other identifiers. However, attackers are not able to break encryption or cryptographic signatures. Being a routing protocol, Sprout does not explicitly address jamming or MAC layer denial of service attacks. However, routes affected by localized jamming or DoS attacks are automatically avoided by Sprout.

Each node holds a unique, signed, public-private key pair, which we will also refer to as the node's *identity*. Valid identities are signed by an offline certifying authority (CA). For the purpose of this presentation, a node's identity is created and signed by the manufacturer, and then permanently stored in the device. Note the similarity to how MAC hardware addresses are assigned: after the manufacturer assigns the identity, anybody may purchase the device. This very lax certification policy avoids all of the problems generally associated with having a CA: it is merely a manufacturing detail. A similar method works for existing devices: identities can be made available for purchase from the CA at some minimum price. The minimum price is a requirement in Sprout, lest an attacker create an unlimited number of identities[3]. An attacker may still purchase or otherwise acquire a large number of identities for use in an attack. In Sec. VI, we evaluate Sprout with respect to large numbers of attacker identities.

It is assumed that a node s communicating with node d has securely acquired the identity of d, or is able to do so on demand. If nodes are to have IP addresses, Sprout can accommodate this through the use of "Statistically Unique and Cryptographically Verifiable" (SUCV) [14] identifiers.

Each node exchanges a symmetric key with each of its known neighbors, using the public key of the neighbor to bootstrap key setup. The symmetric key is used to encrypt all one-hop communication between nodes. Each source also exchanges a symmetric key with any active destination upon connection establishment. This key is used to sign end-to-end acknowledgments, and for end-to-end payload encryption.

Nodes are assumed to have enough computational power to perform verifications of signatures produced by a public key crypto-system, as well as for creating signatures, both at a rate similar to the rate with which the link state changes. In contrast with most previous work, we *do not* assume that the network is $(A+1)$-connected, where A is the number of attackers. However, we do assume that the network can remain connected in the absence of attacker nodes.

B. Constructing the Link State Graph

At the core of any link-state routing protocol lies an algorithm for constructing a link-state graph. In an unprotected link-state protocol, attackers have an unlimited opportunity to pollute the

[3]Clearly, if a public-private key system is already in place, these (more secure) identities can be used in place of the minimum-security identities required by Sprout.

Type	Field Name	Comment
public_key	node1	The ID of Node 1
certificate	node1_cert	Signature (of ID) from the CA
seqno	node1_seq	Sequence Number
public_key	node2	
certificate	node2_cert	*same as above*
seqno	node2_seq	
signature	node1_sign	Signature of msg above line
signature	node2_sign	Signature of msg above line

TABLE I
LINK ESTABLISHMENT MESSAGE

Type	Field Name	Comment
public_key	node_id	The ID of the Node
public_key	neighbor_id	The ID of the lost Neighbor
certificate	node_cert	Signature (of ID) from the CA
seqno	node_seq	Sequence Number
signature	node_sign	Signature of msg above line

TABLE II
LINK TEAR-DOWN MESSAGE

global link-state graph. We propose a link-state construction scheme with several security features to limit the types of fake links that an attacker can introduce:

- Link-state updates are signed and based on the public keys of the two end-points of a link.
- Link-state updates are exchanged between neighbors over links encrypted by efficient symmetric cryptography.
- The TrueLink [15] protocol is used to protect against a "Wormhole attack".

Sec. IV provides a detailed security analysis. TrueLink is a timing-based MAC-layer mechanism that verifies that a direct link exists to an apparent neighbor. It was shown in [15] that TrueLink makes it theoretically impossible (constrained by the speed of light) to create wormholes longer than the nominal transmission radius. Note that while TrueLink effectively stops Wormhole attacks, it does not stop colluding attackers from creating the appearance of links incident upon one or more of the nodes under their control. This capability is sufficient to wreak havoc with most routing protocols, and Sprout was specifically designed to address this problem. We now provide details on our link-state dissemination technique.

When a new link is discovered, or when a previously announced link fails, a link-state update message is broadcast throughout the network. Tables I and II describe the format of the link-state update messages.[4]

The secure broadcasting scheme proposed by Perlman [13] is used to ensure that link updates reach all nodes. Nodes reserve buffer space for each unique and certified public key, and only forward messages with verified signatures. Recall that all one-hop communication is protected and signed through symmetric cryptography. Only link-state updates received from one-hop neighbors with an already established symmetric key are passed to the relatively CPU-intensive verification algorithm. Should a neighbor forward a link-state update which does not pass the signature verification (after passing a standard CRC check), it is marked as faulty or malicious. Any communication from nodes marked as faulty or malicious is ignored. In addition, Sprout limits the rate at which any pair of nodes may issue link-state updates to one update every t_{RATE} seconds. In combination, this effectively mitigates DoS attacks on CPU-intensive cryptographic operations.

[4]The size of a link establishment message, using Elliptic Curve Cryptography [16] is around 128 bytes. A tear-down message is approx. 64 bytes.

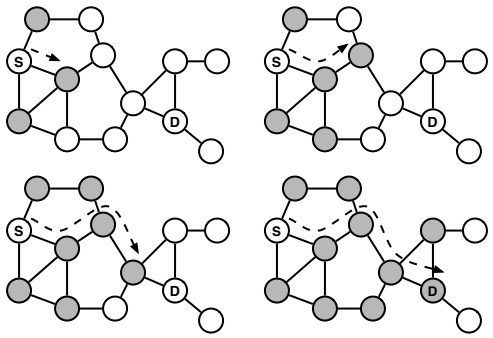


Fig. 1. Successfully generating a simple path through a link-state graph. As the route grows, more nodes (gray) end up in the $considered$ list.

C. Probabilistic Route Generation

Even with the measures introduced above, the link state graph may potentially be polluted by certain classes of non-existing links. A probabilistic route generation algorithm makes Sprout robust to such attacks.

The route generation algorithm probabilistically generates *simple paths* through the link state graph. A *simple path* contains no loops. Moreover, only nodes that are adjacent on the path have direct links between them (i.e. no short-cuts). [5]

To see how a simple path is generated, consider the network depicted in Fig. 1. Route generation starts at node S, with D as the destination. From node S, there are three potential next hops, one of which is chosen uniformly at random. We will call nodes that have already been considered for inclusion $considered$, and the remaining nodes $available$. As an example, once the first hop has been selected, none of the three $considered$ nodes (colored gray) may subsequently appear on the generated route. This includes the selected node, as nodes may only appear once in a simple path.

Algorithm 1 presents the high-level algorithm for the probabilistic generation of a route. This algorithm assumes that the destination node is part of the connected link-state graph. At each step, until the destination is reached, we select a new next-hop and append it to the route. If we reach a point where there is no next hop, and we still haven't reached the destination, we restart the process and try again. All nodes are once again open for consideration. The $considered$ list contains a list of all nodes that have so far been either considered, or chosen, to be the next hop at some point along the route. The function $select\_one$ takes a list and the destination as arguments and returns one element from the list. If the list contains dst, $select\_one$ returns dst. Otherwise, an element is selected from the list uniformly at random.

Each hop of a generated route is probabilistically chosen among the $available$ neighbors of the current router. Since no metrics are used in making the choice, an attacker cannot manipulate this choice by any means other than announcing fake links. As is often the case with randomized algorithms,

[5]Note that in restricting routing to use simple paths, we rely on good nodes to only announce links of "sufficiently high" quality.

Algorithm 1 Function generate_route(src, dst)

```
tries ← 0
while n != dst and tries++ < MAX_TRIES do
  n ← src
  route ← '(), considered ← '(n)
  while n != dst and (n.neighbors - considered) != '() do
    next ← select_one (n.neighbors - considered, dst)
    considered ← considered + n.neighbors
    route ← route + '(next)
    n ← next
  end while
end while
if tries < MAX_TRIES then
  return route // route reached destination
else
  return '() // after MAX_TRIES no route was found
end if
```

the first route generated may be of poor quality. However, given a small number of attempts, the probability of finding a high-quality route is high (see Sec. VI.)

D. Packet Forwarding and Source Route Representation

Each packet contains the entire route from the source to the destination, represented as an array of node identifiers. The header also contains a hop count, which is incremented at each node on the route to the destination, acting as an index into the source route array. Public keys are long and cumbersome to use in a source route. Instead, we apply a secure one-way hash function on each node's public key, and use the hash value in the source route representation.At each intermediate node along the route, the next hash value in the source route is matched against the hash values of the node's neighbors. In the extremely rare event of a hash collision, the packet is forwarded to all matching neighbors. Once the packet reaches its destination, a signed network layer acknowledgment is sent back on the reverse route.

E. Route Selection with Route Performance Feedback

Route generation in Sprout is extremely exploratory, generating routes with no regard to prior history, or expected performance. To ensure good performance, a *route selection* algorithm balances the exploration of new routes with the exploitation of known, active routes of reliable quality.

For every packet transmission, Sprout will either generate a fresh route, using the algorithm described above, or use one of the currently active routes. A fresh route is generated with probability $p(c)$, where c is the number of currently active routes. We use

$$p(c) = max(K, \frac{1}{1+c}),$$

where K is a positive constant < 1. This results in new routes being generated quickly initially, and ensures that new routes are continuously probed throughout the operation of the protocol. Second, if a fresh route was not generated, a route is selected from the set of active routes, with a probability proportional to its score $\sigma(r)$ as described below. In memory constrained environments, a constant upper limit on the number of active routes is applied. When a new route is sampled, this route replaces the lowest-scoring route in the set of active routes.

Before a route can be scored, measurements of route performance are needed. In order to probe the quality of a route, one or more packets are sent along it. Depending on the quality of the route, and the presence of any attackers on the route, we may or may not receive an acknowledgment for the packet. A per-route transmission window keeps track of the last N packets sent, including the time of transmission, and a packet sequence number. The same number is included in the acknowledgment for a packet. This information is used to gather several statistics about the route (listed in Table III.)

r_{out}	Packets sent but not yet acked
r_{rtt}	Round-trip time (exp. avg.)
r_{late}	Packets not yet acked, delay $> \gamma r_{rtt}$
r_{pdr}	Packet delivery ratio (exp. avg.)

TABLE III
PER-ROUTE STATISTICS USED FOR COMPUTING THE ROUTE SCORE.

Upon the receipt of an acknowledgment on route r, any outstanding packets with lower sequence numbers that were sent on route r are considered lost, and their records are removed from r's transmission window. Acknowledgments are returned along the reverse route. Thus, no packet reordering occurs which may otherwise cause packets to be considered lost improperly. r_{out} is the current number of records in the transmission window. The avg. round-trip time, r_{rtt} is calculated using an exponential average, with a tunable parameter α_{rtt} determining the rate of adaptation. The statistic r_{late} is similar to r_{out} but counts only the records where the ack is delayed by more than γr_{rtt}, where $\gamma > 1$. The optimal value of parameter γ depends on the underlying delay characteristics of the network. Values close to 1 are well suited to networks with highly predictable delay, whereas higher values may be necessary where more variance in delay is expected. Finally, r_{pdr} indicates the packet delivery ratio, computed as an exponential average, with a tunable parameter α_{pdr}. Choosing low values for α_{rtt} and α_{pdr} results in a protocol that is highly reactive to recent events, whereas values closer to 1 provide more stability, focusing on the longer term performance of a route.

Given a set of active routes, what routes should we use, and in what proportion? This is determined by the scoring function $\sigma(r)$. Our goal is to strike a good balance between diversity and performance optimization. To compute $\sigma(r)$, we use a heuristic that combines several of the collected statistics:

$$\sigma(r) = \frac{\left(\frac{r_{pdr}}{r_{rtt}}\right)^2}{(1 + r_{late} * r_{out})}. \quad (1)$$

Let us take a closer look at Eq. 1. The term $\frac{r_{pdr}}{r_{rtt}}$ is the inverse of the expected delivery time, which can be seen as a *quality* estimate of the route. By putting the square of the route quality in the numerator, we ensure that route selection will tend toward high-quality routes. In the denominator, outstanding packets give a strong negative bias to a route, but only if 1 or more packets are late (beyond γr_{rtt} ms). By penalizing routes with late packets, route selection quickly shifts away from routes that encounter a sudden drop in performance; yet it does not avoid reliable routes with many outstanding packets.

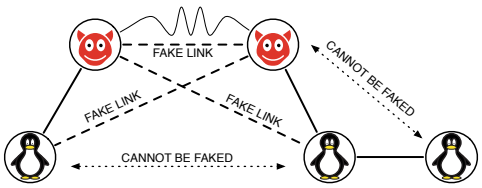


Fig. 2. Solid lines indicate actual links. Attackers (top) cannot fake links between normal nodes (bottom). They are limited to faking links between themselves, and links between an attacker and any neighbor of an attacker.

Note that $generate\_route$ generates routes in a memoryless fashion. Thus, active routes may occasionally be re-generated. This provides a way for routes that experience temporary packet losses to recover from drastically decreased scores.

IV. Analysis - Attack Resilience

Unprotected wireless multi-hop networks are subject to a wide range of attacks. In this section, we analyze two major classes of attacks, pollution of the link-state graph, and dropping of payload packets. In general, an attacker may wish to add links to the link-state graph in an attempt to make most data packets go through it. Once a packet is within the attackers' control, it can choose to drop it, risking detection, or forward it, increasing the probability that additional packets are sent through it in the future. Some aspects of these attacks are difficult to analyze, and we address these through simulation and testbed experimentation, presented in Sections VI and V.

A. Polluting the Link State Graph

Maintaining a consistent view of the network topology is an important component of any link-state routing protocol. A successful attack on the link-state graph will either add a "fake" link, or remove an actual link, from the link-state tables of some or all nodes in the network. Fig. 2 illustrates the types of fake link attacks Sprout can and cannot protect against.

A correct link establishment or tear-down message, see Tables I and II, requires the signatures of one or both end-points of a link. This effectively prevents the attacker from announcing or removing links between two "good" nodes. To prevent an attacker from "tunneling" packets between two good nodes, to make it appear as if they are neighbors (also known as a *wormhole attack*), the TrueLink [15] protocol is employed at the MAC layer. The remaining attacks all involve the addition of "fake" links, or links between attacker nodes. These attacks are exhaustively listed and addressed below.

Adding a Link Between an Attacker Node and a Neighbor of any Other Attacker. If the attacker controls more than one node, it may, through a combination of address spoofing and key sharing, create the impression that one node simultaneously appears in the location of all other attackers. This makes it possible to fool a "good" node into believing it is the neighbor of a remote attacker, whereas it is actually communicating with a nearby colluding attacker.

The attacker may use this technique to increase the degree (number of neighbors in the link-state graph) of the nodes under its control. This increases the expected probability that an attacker node is selected as the next hop in any given step of the path generation process. However, note that the overall degree of the attacker (the total number of nodes with a direct link to one or more attacker nodes) does not increase through this attack. An interesting equilibrium exists where the attacker is unable to add more links without reducing the effectiveness of its attack. A node with very high degree is likely to be considered early on. However, nodes are considered at most once per path generation attempt. Thus, high-degree nodes are likely to quickly be removed from consideration. In addition, after adding a high degree node to the route, the next step will lead to a large number of nodes being marked as considered. This, in turn, increases the probability of the path generation attempt failing. We evaluate the effect of fake link attacks experimentally in Sections VI and V.

Adding a "Virtual" Attacker Identity to the Graph. This attack, sometimes called the *sybil attack*, is similar to the one described above. If an attacker has access to a number of identities (i.e. public-private key pairs signed by the offline CA) larger than the number of physical nodes it controls, it may be able to create the appearance of more than one node existing in a single physical location. This increases the probability that payload packets are sent through attacker nodes. Attacks on payload packets are addressed in IV-B, and the effect of the sybil attack is simulated in Section V.

Adding a Link Between Two Attacker Nodes. Since the attacker controls the cryptographic keys of both end-points, this attack cannot be prevented in the link-state collection stage. Instead, this attack is effectively addressed by the proposed probabilistic path generation technique.

Recall that paths are generated one hop at a time, starting at the source. In each step, one of the *non-considered* immediate neighbors of a node is chosen uniformly at random. Thus, in order for a link between two attackers to be chosen, the path must already include at least one attacker node. Thus, adding links between two attacker nodes does not constitute an effective attack.

Summary. The global link-state graph in Sprout is highly robust to pollution attacks. However, two feasible attacks were identified: a) adding links between an attacker and any neighbor of any other attacker, and b) adding virtual attacker identities to the graph. These attacks are countered by the route generation and selection components, as discussed below.

On the Presence and Effectiveness of Attacker Nodes: Let us assume the attacker does not announce fake links, but simply drops some or all payload packets that arrive at a node under its control. Our goal is to calculate the probability of a generated route containing one or more attackers. Let us first consider a network without attackers. Let N be the number of "good" nodes, and D their average degree. Let L be the average path length of a generated path between a pair of randomly chosen "good" nodes. We can then write the probability of generating a successful path of length k as

$$p(k) = \frac{D}{N}(1-\frac{D}{N})^{k-1}\prod_{i=0}^{k} 1-(1-(1-\frac{D}{N})^{i-1})^{D}. \quad (2)$$

Here, the terms on the left compute the probability of reaching the destination in step k, and the product term is the probability of not reaching a dead end in any step $< k$. The full derivation of this expression is available in [17]. For

$\frac{D}{N} \to 1$, $p(1)$ goes to 1, and $p(k)$ goes to 0 for $k > 1$. This is consistent with our model, since a fully connected network will always find a one-hop path to the destination.

The probability that a path generation attempt is successful can be written as $\sum_{k=1}^{N} p(k)$. Thus, the expected path length E[k], given that the attempt was successful, is:

$$E[k] = \sum_{k=1}^{N} k \frac{p(k)}{\sum_{j=1}^{N} p(j)}. \quad (3)$$

Numerically, with N=200 and D=8, we get a probability of 0.739 of a path generation attempt being successful, and an expected path length of 13.7 hops. Recall that Sprout generates many routes of varying quality, and sends traffic over these paths according to their performance. This means that the expected number of hops traversed by the average packet is generally significantly smaller than the expected length of a freshly generated route.

We can now determine the probability that a given path contains at least one attacker node. Using the same scenario as in the previous subsection, let there be A attacker nodes, chosen uniformly and at random from the original set of nodes. Let the graph, with the attacker nodes removed, remain connected. This preserves pairwise connectivity between all normal nodes in the absence of the attacker nodes, without which it is impossible for any routing protocol to work. The probability of a node chosen at random being an attacker node is $\frac{A}{N}$. Thus, the probability of a given path being compromised, p_c equals one minus the probability of the path containing no attackers, or

$$p_c = 1 - (1 - \frac{A}{N})^{E[k]}.$$

Numerically, with $N = 200$, and $D = 8$, we have $p_c = 0.066$ for $A = 1$, $p_c = 0.129$ for $A = 2$, and $p_c = 0.68$ for $A = 16$. This means that as the number of attackers increases, the probability of generating a compromised route goes up. However, even for a large number of attackers, a significant fraction of generated paths are not compromised. These results match our simulation results well (see Fig. 5).

B. Attacks on Payload Packets

In an open network containing attacker nodes, it is inevitable that some payload packets will be sent on a route that contains an attacker. The attacker then has the choice of forwarding the packet, or dropping it.[6]

We would like to determine the impact of attacker nodes being part of one or more routes in the set of active routes to a destination. Our metric of choice is the probability that a given packet gets dropped by an attacker node. The scoring function in Eq. 1 can not be analyzed easily, due to the highly dynamic nature of the variables r_{late} and r_{out}. However, we can upper bound the impact of an attack by setting $r_{late} = 0$ for all routes for the sake of the analysis. This has the effect of ignoring the dynamic penalty on late (possibly lost) packets, and reduces Eq. 1 to the more tractable $r_{score} = r_{pdr}^2 / r_{rtt}^2$.

[6]Other possibilities include recording the encrypted packet, delaying the packet, observing the timing or simply existence of traffic in the network, etc. Such attacks are outside the scope of this analysis.

Let us first consider a simple scenario, with only two paths. Let both paths have the same round-trip time r_{rtt}, but one of the paths contains an attacker node. Let us assume that both paths have a natural packet delivery ratio of $1 \geq \bar{r}_{pdr} > 0$, reflecting spurious losses. In addition, the attacker node forwards a fraction μ of the packets received, to avoid detection; thus the compromised route has an overall packet delivery ratio of $\hat{r}_{pdr} = \mu \bar{r}_{pdr}$. Given N good paths and M compromised paths, we can write the steady state probability that a given packet is sent over the compromised route as

$$p_q = \frac{\mu^2}{\lambda^2 \frac{M}{N} + \mu^2}, \quad (4)$$

Here λ is a round-trip time factor $0 < \lambda$ which describes the difference in average round-trip-time between "attacker" routes and normal routes. $\lambda < 1$ means "attacker" routes have a shorter rtt than normal paths. Again, the derivation is available in [17]. We can see that in order maximize the probability of dropping packets, the attacker needs to minimize the term $\lambda^2 \frac{M}{N}$, where $\frac{M}{N}$ is closely related to the probability of generating a compromised route, p_c, which was derived earlier. We also see that constructing compromised routes with half the normal round trip time is likely to achieve the same attack power as quadrupling the number of compromised paths. In general, μ_{opt} is the solution to the following cubic equation:

$$\mu^3 + \lambda^2 \frac{M}{N} \mu - 2\lambda^2 \frac{M}{N} = 0. \quad (5)$$

In conclusion, while the attacker can maximize the power of its attack by carefully choosing μ, increasing the number of attacker nodes, or potentially introducing extraordinarily short routes, none of these methods are able to bring the packet delivery ratio close to 0 without an extreme expenditure of resources.

C. Miscellaneous Other Attacks

For completeness, we discuss several additional attacks that have been considered in the secure routing literature. There is no risk of a **blackmail** attack in Sprout. Performance statistics are kept strictly on routes, not nodes, and thus, no opportunity exists for an attacker to make it appear as if a good node is behaving inappropriately. In a **replay** attack, attackers record routing messages, and replay them at a later time in order to cause routing disruptions. In Sprout, sufficiently large serial number fields (32 bits) in link-state updates make replay attacks on control messages, including signed ACKs, infeasible. **Rushing** attacks are specific to reactive routing protocols, and do not apply to Sprout. Lower-layer attacks, such as **Jamming or MAC-Layer DoS** are not explicitly addressed in this paper. The effects of these attacks are localized, as Sprout effectively finds routes around underperforming links. In the **Jellyfish** attack, attackers selectively drop a few packets crucial to upper layers. Due to the small volume of dropped packets, the attack can potentially be mounted without risk of detection by security mechanisms at the lower layers. The use of multiple concurrent paths and randomized path selection in Sprout mitigates this problem. Finally, attackers may reduce

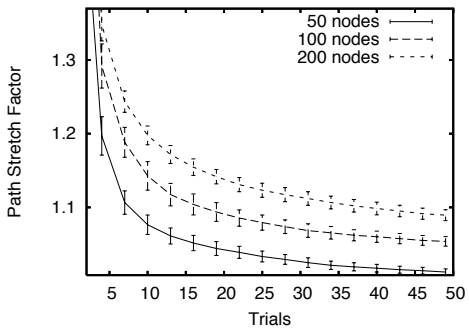


Fig. 3. Path Stretch vs. No. of Trials. Short routes consistently found after several trials. Error bars show 99% confidence interval.

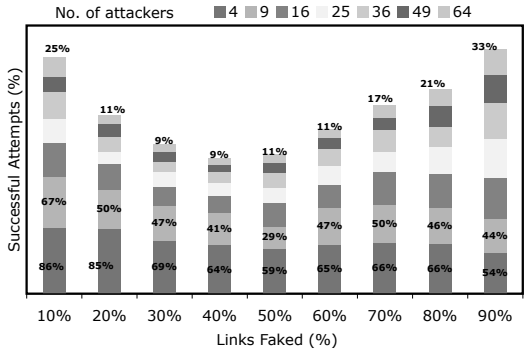


Fig. 4. Successful Attemps vs. No. of Fabricated Links. The route generation algorithm forces attackers to announce fewer than the maximum number of fabricated links.

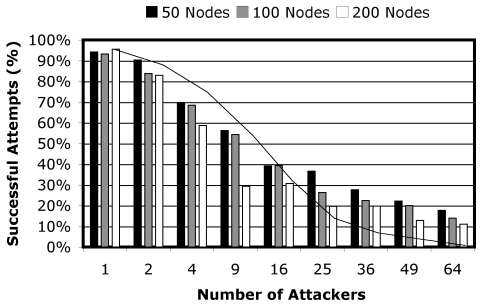


Fig. 5. Successful Attempts vs. Number of Attackers. The line shows analytical results for 200 nodes, for graphs with random, uniform selection of neighbors.

performance for other users by sending large amounts of data. Such problems may be addressed by a variety of fairness mechanisms, which are outside the scope of this paper.

V. Performance Evaluation - Simulation

We use a custom built simulation environment to test the performance and attack resilience of the route generation algorithm in larger networks with many attackers. Path selection and real-world performance is studied in the implementation section. The simulator was built for a high-level study of routing protocols, and does not simulate the MAC/PHY layers. The network graph is generated by placing nodes uniformly at random on a rectangular field. We create edges between nodes that are within transmission range of each other.

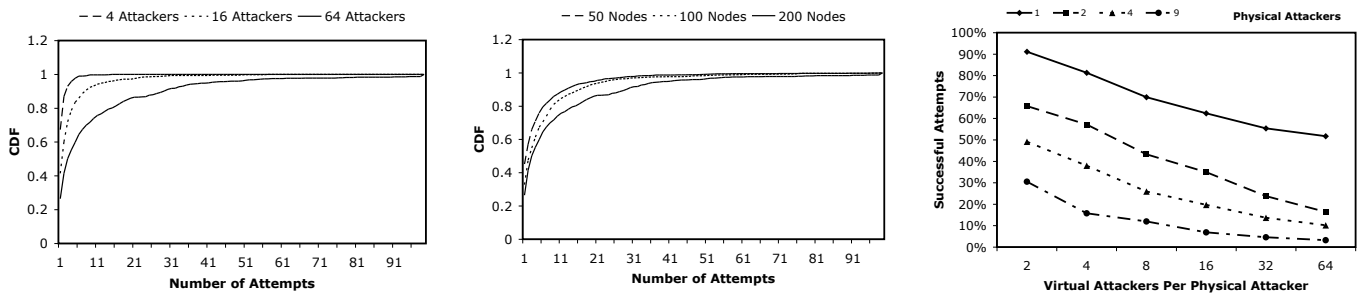
Sprout Routes are Surprisingly Short: To assess the performance of probabilistic route generation, we study the quality of routes generated in larger networks. We select a pair of nodes, and run the route generation algorithm 50 times (trials). This process is repeated 100 times per topology, for 25 topologies, for a total of 120,000 routes generated for each considered network size. We used randomly generated topologies of sizes 50, 100, and 200 nodes. Node density was set to approximately 8 nodes per radio range, which ensured that all graphs were connected. In these experiments, there are no adversarial nodes; the goal of the experiment is to study probabilistic route generation in a benign setting.

Fig. 3 shows the path stretch incurred by Sprout for a varying number of trials. Path stretch is calculated as the minimum length of the generated routes up to a given trial, divided by the length of the route computed by shortest path routing. Results show that the route found in the first trial is on average twice as long as the shortest path. However, as more routes are generated, one can expect to quickly find a route that is within a small margin of the shortest path. For example, with 100 nodes, 15 trials on average finds a path with a length within 10% of the optimal.

All-Out Attacks on Sprout Don't Pay. The following experiments evaluate the resilience of Sprout's route generation algorithm to fake link attacks. We generate 200-node topologies, with between 4 and 64 attackers. The attackers are placed in a grid pattern, optimizing their coverage of the network, whereas the good nodes are placed uniformly at random. The attackers are configured to announce only a percentage of the total number of links that could be faked. These links are chosen uniformly at random. No links between attackers are announced; our analysis suggests that doing so would be a poor attacker strategy, making it less likely that generated routes contain attackers.

Fig. 4 shows the percentage of successful routes found, given a varying percentage of faked links. Each bar combines the success ratio of all attacker scenarios (4-64 attackers), to give an overall picture. We find that to maximize its impact, the attacker should announce only 40 to 50% of the fake links at its disposal. This has to do with a trade-off in the way routes are generated in Sprout, as described in Section III-C. We note that even under ideal attacker conditions, with 200 normal nodes and 64 colluding attackers announcing 40% of the possible fake links, 9% of all route generation attempts result in a good route.

Even Large Numbers of Attackers Cannot Stop Sprout. The strength of an attacker is often measured in terms of the number of nodes under its control. We vary the number of attacker nodes, and measure the success ratio of generated routes in randomly generated topologies of 50, 100 and 200 nodes. Again, node pairs are selected uniformly and at random. Attackers announced 50% of available fake links. Fig. 5 shows that while increasing the number of attackers does result in a consistently lower success ratio, even with 64 malicious nodes attacking a network of 200 good nodes, around 10 percent of route generation attempts result in finding a good route. Recall that once Sprout has found a good route, route selection will make use of it for effective data delivery.



(a) (b) (c)

Fig. 6. (a) CDF of Successful Attempts for varying Number of Attackers (200 good nodes). (b) CDF of Successful Attempts for varying Network Size (64 attackers). For moderate numbers of attackers, routes are found quickly. Even with 64 nodes attacking a network of 200 good nodes, 95% of routes are found within 40 attempts. (c) Successful Attempts vs. Number of Virtual Attackers Per Physical Attacker. With 9 physical attackers and a total of 576 virtual attacker identities, 3% of attempts find a successful route.

For large numbers of attackers, the analytical prediction from Sec. IV is significantly lower than the simulation results. Instead of considering a distribution of path lengths, the analysis considers only the expected path length. In scenarios with large numbers of attackers, long generated routes are very likely to contain one or more attackers. However, shorter routes are still likely to succeed. By only considering the expected path length, the analysis thus provides a pessimistic performance prediction. The simulation captures the distribution of path lengths, resulting in more realistic numbers.

Figures 6 (a) and 6 (b) show CDFs of the number of attempts needed to find the first good route, with respect to the number of attackers, and with respect to network size. Node pairs that are far apart, or that have a particularly difficult network topology between themselves, frequently require a larger number of attempts than nodes that are in the vicinity of each other. We note that 95% of all pairs in the most difficult scenario, find a good route within 40 attempts. Less than 1% of the pairs fail to find a route within 100 attempts.

Sprout Successfully Withstands Massive Sybil Attacks. Although an attacker may control few nodes, it may still be able to acquire a large number of certified public-private key pairs. Using multiple sets of credentials per node, the attacker may be able to increase the strength of its attack. Note that the multiple-identity attack is the strongest version of the sybil attack, where a single node takes on multiple unique identities.

In these experiments, we generate topologies of 200 normal nodes, and vary the number of physical attackers between 1 and 9. Each physical attacker takes on between 2 and 64 identities, and announces fake links for each of these separately. To maximize the effectiveness of the attack, each virtual attacker announces 50% of its available links, selected uniformly and at random. Fig. 6 (c) illustrates the effect of a multiple-identity attack. We observe that a single physical attacker is unable to mount a significant attack; even with 64 virtual attacker identities 60%, of the attempts find a good route, and the number appears to stabilize above 50%. With more physical attackers, fewer attempts are successful. Yet, with 9 physical attackers, each with 64 virtual identities (a total of 576 attacker identities in a network of 200 good nodes), 3% of attempts still find good routes, with 10% of node pairs requiring over 100 attempts to find a good route.

We observe that a massive sybil attack is an effective means of multiplying the effect of an attack. If possible, a mechanism for limiting the number of identities available to an attacker is an effective way of limiting the power of a sybil attack. If such a mechanism is not feasible, Sprout is able to successfully find routes even under a sybil attack of massive proportions. However, finding such a route may take longer than under less adverse conditions. Recall that once a good route has been found, route selection will ensure that the majority of packets are sent along this route. This ensures good overall performance once a valid route to the destination is found.

VI. Performance Evaluation - Implementation

The previous section evaluated the high-level performance of the route generation algorithm. To determine the performance of Sprout in a more practical setting, we implement Sprout in Linux and test it for performance and attack resilience in experiments on our local wireless research testbed.

While we would have liked to compare Sprout against other competitive secure routing protocols, we were unable to find any implementable secure routing protocol theoretically able to withstand the attacks described in this paper, see Sec. II.

In these experiments, we emulate the black hole, gray hole, and fake link attacks. We measure packet delivery ratio, response time and TCP throughput, both under benign conditions and during attack. We demonstrate good general performance numbers and a dramatically increased resilience to attack. For reference, we also show shortest path routing performance. This helps understand the performance penalty incurred by using Sprout for routing security.

Our indoor testbed is comprised of 31 Soekris net4826 nodes, deployed on the 3rd floor of Engineering Building Unit II at the University of California, Riverside; the network is depicted in Fig. 7. Each node runs a Debian v3.1 Linux distribution with kernel version 2.6.13.2 and mounts its root partition over NFS from a server at start-up. We have equipped nodes with EMP-8602-6G 802.11a/b/g WiFi cards, which embed the Atheros AR5006 chipset; the cards are controlled by the latest Linux MadWifi driver [18]. Each card is connected to a 5-dBi gain external omnidirectional antenna. For these experiments, we use the cards in 802.11g mode. TCP selective acknowledgments (TCP_SACK) were enabled.

Our Sprout implementation is based on a combination of the OLSR [19] implementation from olsr.org, for gathering

Fig. 7. Our indoor–testbed deployment. Nodes are represented by dots along with their IDs.

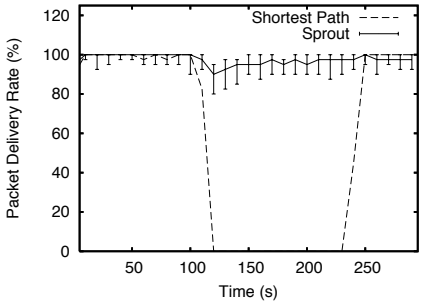


Fig. 8. Median Packet Delivery Ratio vs. Experiment Time. Error bars are 5th and 95th percentile. Black-hole attack starts at time 120s, ends at 240s. Sprout delivery ratio is marginally affected by a black-hole attack.

and distributing topology information, and the Click modular router platform [20], version 1.5, for all Sprout specific functionality, including source routing. The source code for our Sprout implementation is available at [21]. Our implementation also supports shortest path routing, for comparison purposes. All OLSR optimizations for efficient broadcast were turned off, as they are incompatible with the secure broadcast scheme employed, see [13]. Unless otherwise noted, experiments were repeated at least 25 times for reliability of results. In the experiments on the testbed, routes used by Sprout ranged in length between 2 hops and 9 hops. For these experiments, we used $\alpha_{rtt} = 0.9$, and $\alpha_{pdr} = 0.9$, reflecting a tradeoff between long-term stability and near-term "reactiveness to change". We used $\gamma = 1.25$ to allow for a moderate level of variability in the round-trip time.

Sprout is Nearly Unaffected by Black Hole Attacks: In a black hole attack, a node drops all data packets it receives, instead of forwarding them toward their intended destination. Our first experiment involves a single pair of nodes, nodes 20 and 34, and a simple black hole attack mounted by node 31. We set up node 20 to ping node 34 four times per second, and observe the resulting packet delivery ratio (PDR) and round-trip times (RTT). Figs. 8 and 9 show the evolution over time,

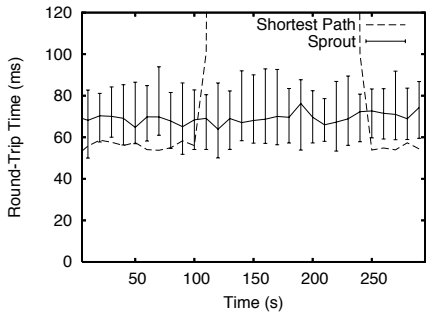


Fig. 9. Median Round-Trip Time vs. Experiment Time. Sprout round-trip time is marginally affected by a black-hole attack.

of the PDR and RTTs achieved by Sprout as compared to shortest path routing. As the black hole attack commences (at time 120s), Sprout experiences a slight drop in PDR, but quickly detects the attack and shifts traffic to better routes. Shortest path regains its performance after the attack ends (at time 240s). Node 31 was, for most of the time, on the shortest path, which is apparent in the shortest path results. A higher average RTT is experienced with Sprout (by about 15 percent) due to route diversity; multiple routes are used, some of which have a longer RTT than the shortest path.

Gray Hole Attacks Don't Work Against Sprout: In a gray hole attack, the attacker drops some, but not all packets. By forwarding some packets, the attacker may give the appearance of operating properly, thereby convincing the routing protocol to send more packets through the attacker. In this experiment, node 31 is once again the attacker, this time mounting gray hole attacks with a packet forwarding ratio between 10 and 90 percent.

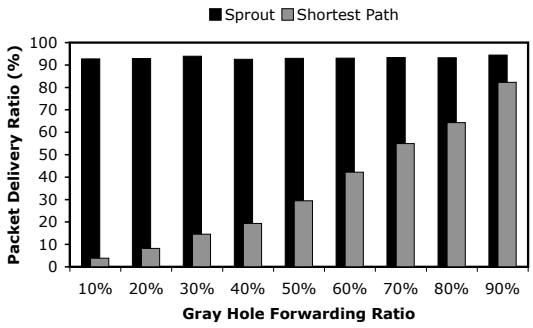
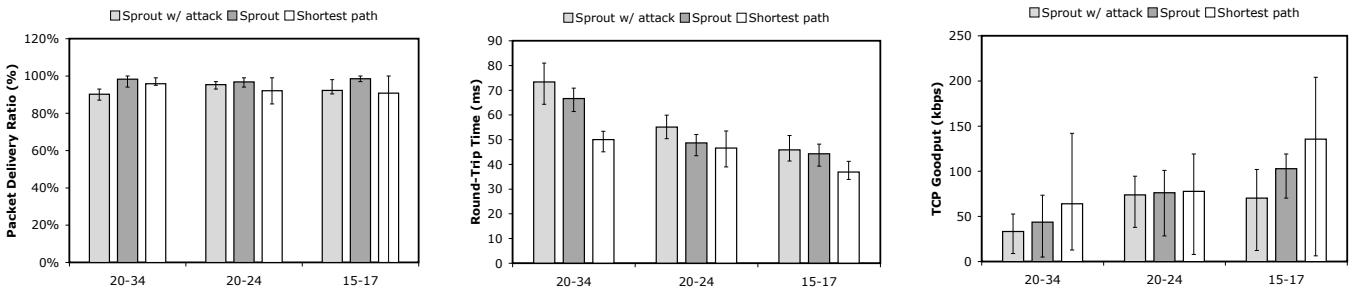


Fig. 10. Packet Delivery Ratio vs. Gray Hole Forward Ratio. Gray hole attack is ineffective on Sprout. Shortest path PDR reflects packet drops on the forward and reverse paths combined.

Fig. 10 shows the end-to-end packet delivery ratio as we vary the percentage of packets forwarded by the gray hole attacker. The route selection mechanism in Sprout detects the high number of packet drops occurring on routes through the gray hole attacker, and avoids these routes, effectively mitigating the attack. We show shortest path results for reference. Shortest path routing incurs packet drops on both the forward and the reverse path, resulting in a measured PDR of approximately p^2, where p is the probability of the gray hole attacker forwarding a packet.

Sprout Successfully Withstands Fake Link Attacks: With



(a) Packet delivery ratio for three pairs. (b) Round-trip times for three pairs. (c) TCP Goodput for three pairs.

Fig. 11. Error bars show the 5th/95th percentile. Shortest path results for the attack scenario not shown: no packets were delivered.

the fake link attack, two or more attackers collude to create fake link announcements for links between distant nodes. Fake link attacks are extremely effective against shortest path and shortest multi-path routing algorithms, as long links between distant nodes appear as valuable shortcuts. In Sprout, attackers are limited to announcing fake links between themselves, and between an attacker and a neighbor of any other attacker.

In these experiments, nodes 21, 27, and 40 were set aside as attackers. A total of 40 fake links were announced with nodes 15, 18, 20, 24, 25, 29, 34, 35, 37 and 41 as endpoints, for those attackers that did not have valid links to these nodes. Attackers were configured to drop all incoming data payload packets, and to forward all link-state updates. Each experiment included four scenarios: a) Sprout with attack, b) Sprout without attack, c) shortest path with attack, and d) shortest path without attack. In the "no attack" scenarios, the attackers did not participate in the network. For each experiment, we performed two measurements. First, a 2-minute ping session, at 4 pings per second. Second, a 1-minute TCP session. We ran these experiments for three node pairs: 20-34, 20-24 and 15-17. We ran these experiments 25 times back-to-back.

Figures 11 (a) through 11 (c) show the results of this experiment. Sprout was highly competitive with respect to the end-to-end packet delivery ratio, consistently outperforming shortest path under benign conditions, and being marginally affected by the attack. Shortest path results are not shown for the attack scenario, as no packets were successfully delivered in this case. As observed earlier, Sprout does incur an increase in round-trip time, as compared to shortest path routing, see Fig. 11 (b). This is due to the higher route diversity. Finally, the TCP results in Fig. 11 (c) shows Sprout providing reliable performance under attack as well as in benign conditions. We note that while shortest path achieves higher average throughput under no-attack conditions, the variability is higher.

VII. Conclusion

In this paper, we present Sprout, a secure routing protocol resilient to a wide range of routing layer attacks, and the first to provide robustness to large numbers of colluding attackers. Unlike previously proposed approaches, Sprout probabilistically generates a multiplicity of routes, without requiring these paths to be node- or edge-disjoint. Each route is continuously appraised by means of signed end-to-end acknowledgements, and the amount of traffic forwarded on a given route is determined by the assessed quality of the route. We demonstrate through analysis, simulation and real-world experiments on our 31-node experimental multi-hop wireless network, that Sprout effectively handles a rich set of possible attacks. We also find that Sprout TCP throughput comes within 15% of that of shortest path routing, a small price to pay for vastly improved security.

References

[1] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina, "SPROUT: P2p routing with social networks.," in *EDBT Workshops*, 2004.

[2] Y-C. Hu, D. B. Johnson, and A. Perrig, "Sead: Secure efficient distance vector routing in mobile wireless ad hoc networks," in *WMCSA*, 2002.

[3] Yih-Chun Hu, Adrian Perrig, and David B. Johnson, "Ariadne: A secure on-demand routing protocol for ad hoc networks," in *MobiCom*, 2002.

[4] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," in *Proc. CNDS*, 2002.

[5] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Mobile Computing and Networking*, 2000.

[6] P. Papadimitratos and Z. Haas, "Secure link state routing for mobile ad hoc networks," in *IEEE CS Workshop on Security and Assurance in Ad hoc Networks*, 2003.

[7] B Awerbuch, D Holmer, C Nita-Rotaru, and H Rubens, "An on-demand secure routing protocol resilient to byzantine failures," in *WiSE*, 2002.

[8] K. Sanzgiri, B. Dahill, B. N. Levine, C. Shields, and E. M. Royer, "A secure routing protocol for ad hoc networks," in *ICNP*, 2002.

[9] B Awerbuch, D. Holmer, and H. Rubens, "Provably secure competitive routing against proactive byzantine adversaries via reinforcement learning," Tech. Rep., Johns Hopkins University, 2003.

[10] Lili Qiu, Yang Richard Yang, Yin Zhang, and Haiyong Xie, "On self adaptive routing in dynamic environments - an evaluation and design using a simple, probabilistic scheme," in *ICNP*, 2004.

[11] V. Borkar and P. Kumar, "Dynamic cesaro-wardrop equilibration in networks," *IEEE Transactions on Automatic Control*, Mar 2003.

[12] Panagiotis Papadimitratos and Zygmunt J. Haas, "Secure data transmission in mobile ad hoc networks," in *WiSe*, 2003.

[13] Radia Perlman, *Network layer protocols with byzantine robustness*, Ph.D. thesis, Massachusetts Institute of Technology, 1988.

[14] G. Montenegro and C. Castelluccia, "Statistically unique and cryptographically verifiable (sucv) identifiers and addresses.," in *NDSS*, 2002.

[15] J. Eriksson, S. V. Krishnamurthy, and M. Faloutsos, "Truelink: A practical countermeasure to the wormhole attack in wireless networks," in *ICNP*, 2006.

[16] A.J. Menezes, *Elliptic curve public key cryptosystem*, Kluwer Academic Publications, 1993.

[17] Jakob Eriksson, Michalis Faloutsos, and Srikanth Krishnamurthy, "Routing amid colluding attackers," http://networks.cs.ucr.edu, 2006.

[18] "The madwifi driver," madwifi.org.

[19] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *IEEE INMIC*, 2001.

[20] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Computer Systems*, August 2000.

[21] Jakob Eriksson, "Sprout source code," networks.cs.ucr.edu/downloads.